AN FPGA IMPLEMENTATION OF ADAPTIVE LINEARIZATION OF POWER AMPLIFIERS

A Thesis
Presented to
The Academic Faculty

By

Sehej Ahluwalia

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2019

AN FPGA IMPLEMENTATION OF ADAPTIVE LINEARIZATION OF POWER AMPLIFIERS

Approved by:

Dr. James Kenney, Advisor
School of ECE
Georgia Institute of Technology

Dr. Richard Causey
School of ECE
Georgia Tech Research Institute

Dr. David Citrin
School of ECE
Georgia Institute of Technology

Date Approved: July 24, 2019

Its a magical world, Hobbes, ol' buddy ... lets go exploring!

Bill Watterson, Calvin and Hobbes

ACKNOWLEDGEMENTS

Dedicated to Manmohan Singh Sawhney; the wiset, strongest and most loving man I'll ever know.

I want to thank Dr. James Kenney for all of his guidance and advice throughout this project, his expertise made all of this possible. Thank you to Dr. Richard Causey and Dr. David Citrin for serving on my committee and for their constant support. Special thanks to Dr. Timothy Brothers for his outstanding mentorship and uncanny ability to answer any question I had. To my parents, without whom I'd be completely lost, thank you for motivating me and providing me with the opportunities to succeed. As for the rest of my family, friends and professors, I will always be grateful for all that you've done.

Effort sponsored by the U.S. Government under Other Transaction number W15QKN-15-9-1004 between the NSC, and the Government. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Power amplifiers (PAs) are inherently nonlinear devices; utilized in essentially all communication systems around the world. An ideal, linear, PA would take an input signal and simply provide a constant gain; resulting in minimal distortion of the input. However, due to their nonlinear characteristics at higher input powers, these amplifiers end up producing a higher power output that is not a perfect amplified version of the input. Which can lead to information loss in communication systems. Digital predistortion is a popular method of reducing the nonlinear effects of these devices. The algorithm works by modifying the incoming signal before it reaches the amplifier. This modification is designed in such a way that the nonlinear effects are cancelled out after the signal is sent through the PA. Due to the constant changing nature of power amplifiers these algorithms need to be adaptive as to guarentee performance over various conditions.

The approach discussed in this thesis develops a gradient descent direct learning architecture that is implemented entirely in the fabric of an FPGA. The proposal attempts to utilize the parallel computing power of an FPGA along with their high clock and data rates, in order to create a faster adaptive system. The effectiveness of this development is demonstrated via simulations in MATLAB, Simulink and VHDL. The algorithm shows more accurate and more stable results than other predistorter approaches as well as faster adaptive convergence times. The results and comparisons of the various implementations are discussed in this work. Continued development of this approach could allow for faster more robust digital predistorter implementations in a variety of applications.

CHAPTER 1

INTRODUCTION AND BACKGROUND

Power amplifiers (PAs) are inherently nonlinear devices; utilized in essentially all communication systems around the world [1]. An ideal, linear, PA would take an input signal and simply provide a constant gain; resulting in minimal distortion of the input. However, due to their nonlinear characteristics at higher input powers, these amplifiers end up producing a higher power output that is not a perfect amplified version of the input. Which can lead to information loss in communication systems [2]. Figure 1.1 from [3] shows this problem:

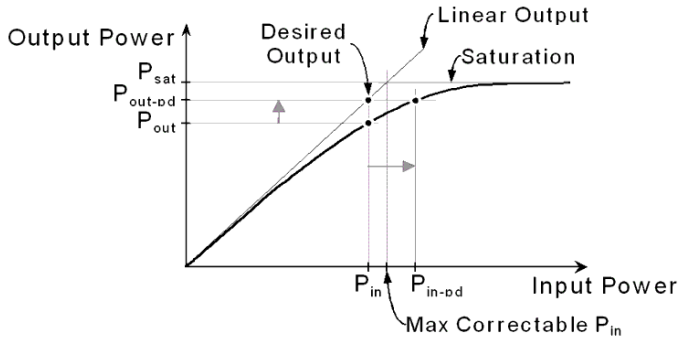

Figure 1.1: Characteristic Nonlinear Behaviour of a PA

There are two main solutions to this issue. The first, and probably most obvious one, is to simply operate the amplifier at lower input power, such that the behavior is in the linear region [4]. While this method does work, PAs are more power efficient at higher input power levels; meaning you are trading operational power for performance. The other method is known as Digital Predistortion (DPD), which acts as an inverse to the nonlinear behavior of the device. This generates the desired behavior at higher input power levels [2]. However, there are limits to this corrections effectiveness. After a certain point the PA will saturate and no amount of predistortion could generate the desired behavior. The figure below demonstrates the operating regions for both approaches [3]:

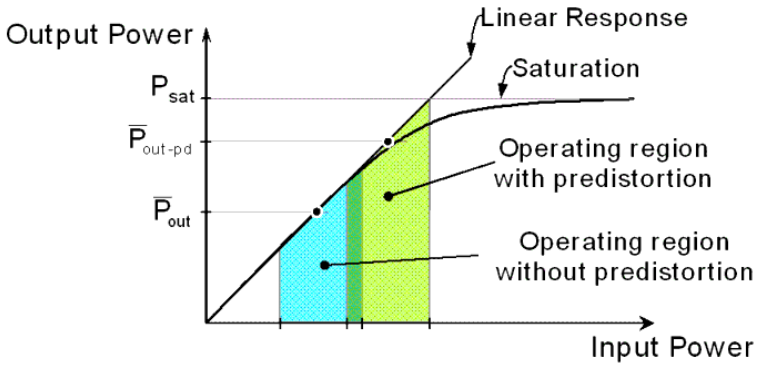

Figure 1.2: Operating Regions With and Without DPD

It can be seen that the performance of predistortion is dependent on the Output Back-Off (OBO) defined to be the relation of the output saturated power of the PA relative to the mean power of the signal [5].

$$OBO = 10\log_{10}\left(\frac{P_{sat}}{P_o}\right) \tag{1.1}$$

It can be seen that the higher the OBO, the more linear the PA. The effectiveness of the DPD algorithm is typically measured in Nominal Mean Squared Error (NMSE) expressed in dBs. NMSE is given by the following formula:

$$NMSE = 10\log_{10}\left(\frac{\sum_{k=1}^{K}\left|d_k - \hat{d}_k\right|^2}{\sum_{k=1}^{K}|d_k|^2}\right) \tag{1.2}$$

Where K are the number of samples collected, d_k is the ideal linear output of the PA, and $\hat{d}_k$ is the real output of the device with the predistortion algorithm running. Typical values of NMSE are around -40 dB, but can range from -25 dB to -55 dB depending on the OBO [6].

1.1 Digital Predistortion

In order to develop a predistorter algorithm, an estimate of the nonlinear PA function needs to be determined. Due to the internal physics of some amplifiers, this nonlinear function must also take into account the memory effects of the device. The current output is not only a nonlinear function of the current input, but also depends on previous input values up until the estimated memory length of the PA [7].

Traditionally, these PA nonlinearities are modelled using a Volterra series; a mathematical model that captures the nonlinear terms and memory effects present in these devices [8]. A finite Volterra Series of the input $x(n)$ with power P and memory length N is written as:

$$y(n) = \sum_{p=1}^{P} y_p(n) \tag{1.3}$$

Where

$$y_p(n) = \sum_{i_1=0}^{N-1} ... \sum_{i_p=0}^{N-1} h_p(i_1, ..., i_p) \prod_{r=1}^{p} x(n - i_r) \tag{1.4}$$

$h_p(i_1, ..., i_p)$ is referred to as the Volterra Kernel. To correctly model the PAs, the values of the Volterra Kernel's must be found such that the Volterra model most closely matches the actual behavior of the device.

It can be seen that this formulation quickly becomes complicated as the order is increased, with any order greater than 5 being very difficult to solve [9]. As a result, research has been done to simplify the Volterra series, making the trade off between computing speed and accuracy. Current solutions to this issue is memory polynomial, a simpler version of the Volterra series and the Dynamic Deviation-Reduction Volterra series model[9].

The predistortion block can be thought of as an inverse to this Volterra series, resulting in a linear output from the PA. Predistortion algorithms work to find the Volterra coeffi-

cients for the DPD block that minimize the error between the PA output and the desired signal. Figure 1.3 provides a visual depiction of DPD and how it is used to get the ideal behavior from a nonlinear device, from [10]:

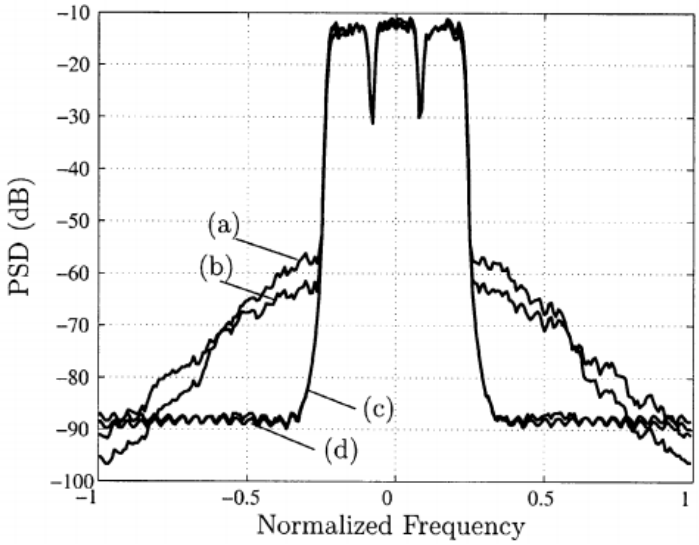

Figure 1.3: Example of DPD

In Figure 1.3, (a) PA output with no DPD. (b) PA Output with DPD that does not account for memory effects. (c) Output with full DPD. (d) Ideal Output.

1.1.1 Direct and Indirect Learning

The two main approaches to DPD design and implementation can be split into direct and indirect forms of learning [11]. The block structure of the two systems are shown below:

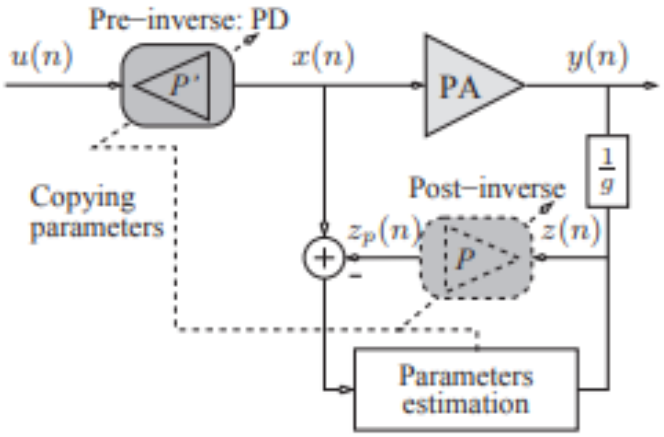

Figure 1.4: Indirect Learning Model

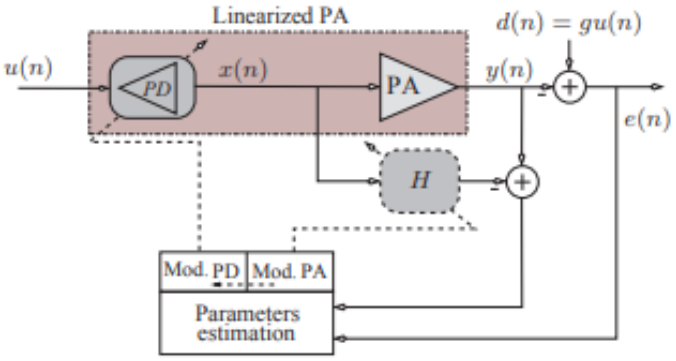

Figure 1.5: Direct Learning Model

The indirect learning structure attempts to find the model that best fits the output of the PA to its input, this is known as a postdistorter. The postdistorter is then assumed to be equal to the predistorter,with cetain restrictions as discussed in [2]. In this case, the parameters are copied into the DPD block for use. Direct learning first models the PA itself and the uses that model to find a true inverse of the PA for the DPD block. It has been shown that direct learning models perform better and have lower numerical instability than indirect implementations [11, 12, 13]. Due to this fact, a direct learning predistorter architecture was chosen to maximize performance, despite its more complex structure.

Another common direct learning architecture is shown in Figure 1.6, where there is

no PA model calculation used for the predistorter. In the proposed gradient descent implementation, knowledge of the PA nonlinear function is incredibly important for accurate cost function calculation. As a result, the more complex structure in Figure 1.5 was chosen.

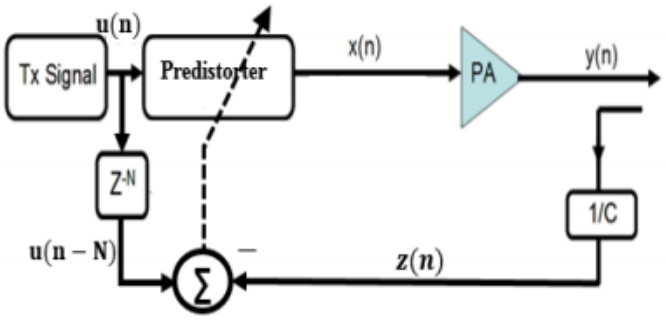

Figure 1.6: Alternate Direct Learning Model

With newer work in this field, lower complexity direct learning models have also been developed, making them even more attractive of an implementation [14].

Predistortion is considered an adaptive algorithm as the estimates of the Volterra coefficients are calculated in real time as data is being run through the device. The nonlinear behavior of the PA can change with time, due to temperature and other factors. Therefore the estimation must be adaptive in order for ideal performance to be maintained, even after initial convergence [15].

1.1.2 Current Presdistortion Implementations

Currently, DPD is split into two separate parts. First, the model and inverse calculation are implemented on a Digital Signal Processing (DSP) chip. Second, the predistortion itself is a hardware implementation using look up tables on a Field-Programmable Gate Array (FPGA) [15]. This allows for the more computationally intensive processes to be done in the background at lower speed, updating the look up tables periodically [16]. A visualization of this architecture is shown in Figure 1.7.

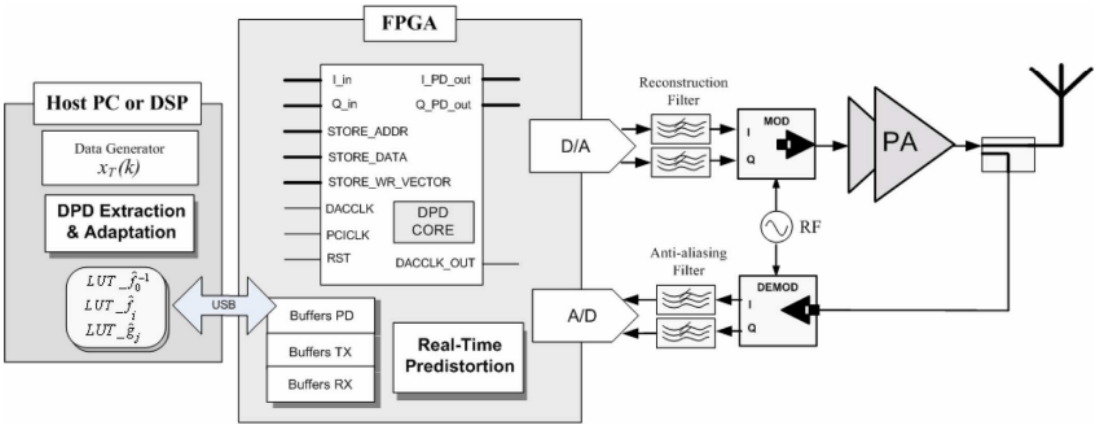

Figure 1.7: Traditional Predistortion Architecture

More recent implementations of DPD blocks have been focusing on the idea of subsampling to deal with larger bandwidth requirements [17]. Predistortion requires a feedback path in order to operate and forthcoming systems may require signal bandwidths of more than 1 GHz. Subsampling allows for operation of the predistorter with feedback sampling occurring below the input Nyquist frequency. Another solution to the bandwidth issue is to use high speed sampling Analog to Digital Converters (ADCs) which are becoming more commonplace in new technologies [18].

With increasing data rate and bandwidth requirements coming from newer communication systems, predistortion algorithms have needed to adapt, becoming faster and more wide band than ever before [19, 20]. The proposed FPGA implementation has the potential to increase the speed of adaption by utilizing the speed of FPGA platforms while also keeping in line with bandwidth requirements by utilizing high speed samplers. While some pure FPGA implementations have been published [21, 22, 23] they utilize indirect learning techniques that have been shown to be mathematically unstable solutions. Since DPD performance is heavily reliant on the PA and the input signal, comparison across literature can be difficult to do fairly. As a result, the performance of the proposed design will be compared with other self-implemented predistortion algorithms using the same input signal

and nonlinear model.

CHAPTER 2
TECHNICAL APPROACH

The goal of this research was to generate and simulate a pure hardware implementation of a digital predistortion utilizing an adaptive direct learning approach. As previously stated, traditional DPD utilizes an FPGA for the data path implementation, but the actual learning algorithm takes places on a DSP block or processor. If the entire system was running on the FPGA fabric there would be no latency for parameter updates which could result in faster adaption times [21].

2.1 Field-Programmable Gate Arrays

A Field-Programmable Gate Array or FPGA is essentially a configurable integrated circuit. They are widely used in signal processing and communication systems. As they are hardware implementations of various functions, they have the benefit of parallel processing, where multiple operations can be executed concurrently [24]. The trade-off to higher processing speed, is that compared to microcontrollers. FPGAs are unable to perform more complex operations efficiently, such as large matrix inversions. So while FPGAs can perform parallelized real-type data processing, software systems are more powerful and varied in the tasks they can perform.

In order to make use of the high speed of an FPGA platform, care must be taken when designing the algorithm to ensure it will work with the platform. As a result, a memory polynomial direct learning approach was taken utilizing gradient descent to balance the predicted results with computational complexity. Some implementations have attempted to use neural networks for predistorters as shown in [25], but these implementations are much more complex and are not efficiently translated to FPGA fabric [26]. Memory polynomials have been widely used in DPD devices and gradient descent algorithms have been shown

to work well in FPGA fabric [27].

2.2 Memory Polynomials

The memory polynomial is a simplified version of the Volterra series that still effectively captures memory effects. The memory polynomial model with the input $x(n)$ with order K and memory length Q is written as [12]:

$$y(n) = \sum_{k=0}^{K} \sum_{q=0}^{Q} a_{kq} x(n-q)|x(n-q)|^k \tag{2.1}$$

Where a_{kq} is the polynomial coefficient for a specific delay and order. As with a general Volterra series, the memory polynomial is linear with respect to its coefficients, which is extremely useful when it comes to coefficient estimation.

Defining the basis function $\phi_{kq}[x(n)] = x(n-q)|x(n-q)|^k$ we can rewrite the memory polynomial as:

$$y(n) = \sum_{k=0}^{K} \sum_{q=0}^{Q} a_{kq} \phi_{kq}[x(n)] \tag{2.2}$$

Finally, defining the vectors $\boldsymbol{x}(n)$ and $\boldsymbol{a}(n)$ we get the following result:

$$\boldsymbol{x}(n) = [\phi_{00}[x(n)] \;\; \phi_{10}[x(n)] \;\; ... \;\; \phi_{K1}[x(n)] \;\; ... \;\; \phi_{KQ}[x(n)]]^T \tag{2.3}$$

$$\boldsymbol{a}(n) = [a_{00} \;\; a_{10} \;\; ... \;\; a_{K1} \;\; ... \;\; a_{KQ}]^T \tag{2.4}$$

$$y(n) = \boldsymbol{x}^T(n)\boldsymbol{a}(n) \tag{2.5}$$

If we are given N samples of input and output data that we wish to fit to a memory polynomial model, we can generate the following linear system:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{a} \tag{2.6}$$

Where $\boldsymbol{y} = [y(0) \quad ... \quad y(N-1)]$ and $\boldsymbol{X} = [\boldsymbol{x}^T(0) \quad ... \quad \boldsymbol{x}^T(N-1)]$. It can be seen that we want to find the coefficients $\boldsymbol{a}$ that best fit the relationship between the given data. Using a least-squares approach the estimate $\hat{\boldsymbol{a}}$ is calculated as [28]:

$$\hat{\boldsymbol{a}} = \left(\boldsymbol{X}^H \boldsymbol{X}\right)^{-1} \boldsymbol{X}^H \boldsymbol{y} \tag{2.7}$$

If the computation is meant to be adaptive, the algorithm can be iterated over multiple data collection cycles. While there are methods such as orthogonal basis that can be better suited for a least-squares estimate, matrix inversion is not guaranteed to be stable. Furthermore an ill-suited matrix could result in no solution [29]. Stability issues, along with the greater computational complexity of a least-squares approach, is why a gradient descent approach was chosen for parameter estimation.

2.3 Gradient Descent Algorithms

Gradient descent is a fairly straightforward method in machine learning. The concept is to minimize the cost function, defined in terms of the parameter you wish to estimate. By calculating the derivative of the cost function J given your current estimate, the idea is to move in the direction of the negative gradient. Thus resulting in a parameter estimate that provides the smallest cost [30]. This is shown through a graph in [31]:

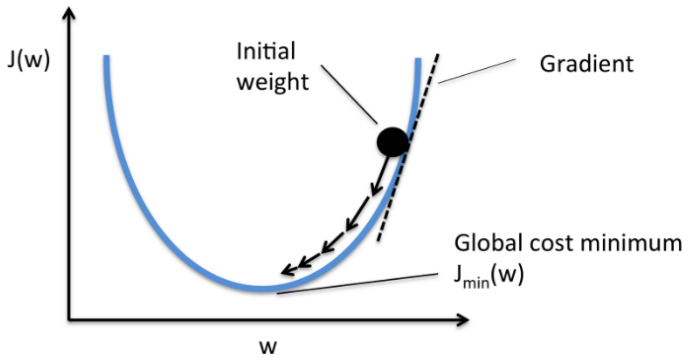

Figure 2.1: Depiction of a Gradient Descent Algorithm

The gradient descent algorithm is largely dependent on the update step size. Too large of a step and the algorithm could be unstable. Too small of a value could cause the algorithm to get stuck in local minima rather than the global minimum of the cost function.

Returning to the estimation problem with memory polynomials, we can see how the parameters can be found using gradient descent. The estimated output at the current sample is written as $\hat{y}(n)$ given the current parameter estimate $\hat{\boldsymbol{a}}(n)$.

$$\hat{y}(n) = \boldsymbol{x}^T(n)\hat{\boldsymbol{a}}(n) \tag{2.8}$$

Defining the cost function as the squared error between the actual value and estimate:

$$J(\hat{\boldsymbol{a}}(n)) = |y(n) - \hat{y}(n)|^2 = \left|y(n) - \boldsymbol{x}^T(n)\hat{\boldsymbol{a}}(n)\right|^2 \tag{2.9}$$

The parameter update equation can then be written as:

$$\hat{\boldsymbol{a}}(n+1) = \hat{\boldsymbol{a}}(n) - \boldsymbol{\mu}\frac{\partial J(\hat{\boldsymbol{a}}(n))}{\partial \hat{\boldsymbol{a}}(n)} \tag{2.10}$$

Where μ is the learning rate and $\frac{\partial J(\hat{\boldsymbol{a}}(n))}{\partial \hat{\boldsymbol{a}}(n)}$ is the derivative of the learning rate with respect to the current parameter estimate. The algorithm will iterate over the data until the error converges to a minimum. It is important to note that since $\hat{\boldsymbol{a}}(n)$ is a vector, the learning rate can also be written as a vector, in case different parameters require different step sizes.

What makes gradient descent for DPD systems different than traditional implementations, is the use of I/Q or complex data in FPGAs. Therefore, gradient descent with complex numbers must be performed.

2.3.1 Complex Gradient Descent

Complex gradient descent is not too much different than the traditional case, in fact, one method of complex gradient descent is to simply split the parameters into their real and

imaginary parts and perform separate convergence calculations on each.

However the work done in [32] and [33] provides a more elegant solution and some new mathematical formulations for derivatives of complex numbers. The gradient of a cost function $J(\hat{\boldsymbol{a}}(n))$ with complex parameters $\hat{\boldsymbol{a}}(n)$ is written as:

$$\nabla = \frac{\partial J(\hat{\boldsymbol{a}}(n))}{\partial \hat{\boldsymbol{a}}^*(n)} \tag{2.11}$$

Where $\hat{\boldsymbol{a}}^*(n)$ is the complex conjugate. Defining a function $g(z)$ where z is a complex number $z = x + iy$ and $z^* = x - iy$, the following definitions also hold:

$$\frac{\partial g}{\partial z} = \frac{1}{2}\left(\frac{\partial g}{\partial x} - i\frac{\partial g}{\partial y}\right) \tag{2.12}$$

$$\frac{\partial g}{\partial z^*} = \frac{1}{2}\left(\frac{\partial g}{\partial x} + i\frac{\partial g}{\partial y}\right) \tag{2.13}$$

$$\frac{\partial z}{\partial z} = \frac{\partial z^*}{\partial z^*} = 1 \tag{2.14}$$

$$\frac{\partial z}{\partial z^*} = \frac{\partial z^*}{\partial z} = 0 \tag{2.15}$$

While implementations in [11] and [6] use a gradient descent approach, they do not utilize the full definition of a complex gradient descent function in their design.

2.4 Direct Learning Implementation

As stated previously, the approach chosen is a direct learning implementation. This means that first, the PA must be characterized and second, the ideal DPD coefficients are found. These are two separate estimation algorithms that must be performed.

The full data chain can be described as two memory polynomials back to back. The first is the predistorter, which feeds into the second polynomial, the power amplifier. The block diagram in Figure 2.2 re-iterates this structure:

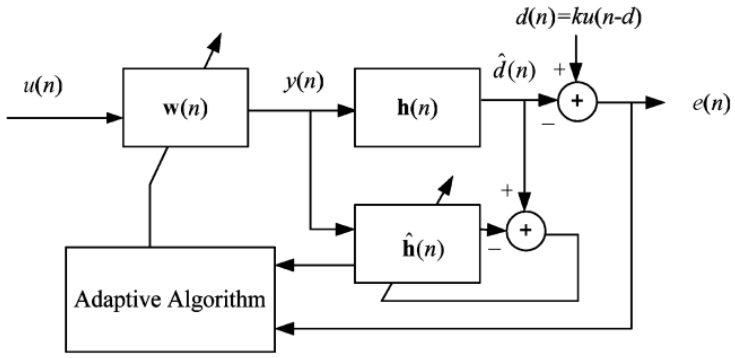

Figure 2.2: Direct Learning Block Diagram

The amplifier's nonlinear coefficients $\boldsymbol{h}(n)$ are unknown and therefore must be estimated, represented by $\hat{\boldsymbol{h}}(n)$. If we define the predistorter as a memory polynomial with order K, and memory length Q, the power amplifier model as a memory polynomial with order P, and memory length Q, the entire data chain can be written as shown below:

$$y(n) = \sum_{k=0}^{K} \sum_{q=0}^{Q} w_{kq} \phi_{kq}[u(n)] \tag{2.16}$$

$$y(n) = \boldsymbol{u}^T(n)\boldsymbol{w}(n) \tag{2.17}$$

$$\hat{d}(n) = \sum_{p=0}^{P} \sum_{m=0}^{M} h_{pm} \phi_{pm}[y(n)] \tag{2.18}$$

$$\hat{d}(n) = \boldsymbol{y}^T(n)\boldsymbol{h}(n) \approx \boldsymbol{y}^T(n)\hat{\boldsymbol{h}}(n) \tag{2.19}$$

Where $u(n)$ is the input signal, $\boldsymbol{w}(n)$ are the predistorter weights, $y(n)$ is the predistorter output, $\hat{\boldsymbol{h}}(n)$ are the estimated PA coefficients and $\hat{d}(n)$ is the output of the PA. The ideal output $d(n) = G * u(n)$ is some linear gain (G) times the input, the goal of the overall system is to drive the error between $\hat{d}(n)$ and $d(n)$ as small as possible. The estimation of $\hat{\boldsymbol{h}}(n)$ is referred to as PA characterization and the calculation of $\boldsymbol{w}(n)$ is denoted as predistorter parameter estimation. The derivations are based on work done in [6] but expanded

upon with the complex gradient definition.

2.4.1 PA Characterization

In order to estimate the amplifier coefficients we define the cost function as the squared error between the actual data from the power amplifier and the estimate of our current model. The error is defined as $e(n) = \hat{d}(n) - \bar{d}(n)$.

$$J(\hat{\boldsymbol{h}}(n)) = |e(n)|^2 = \left|\hat{d}(n) - \bar{d}(n)\right|^2 = \left|\hat{d}(n) - \boldsymbol{y}^T(n)\hat{\boldsymbol{h}}(n)\right|^2 \tag{2.20}$$

At this point the predistorter is disabled, so the input signal passes through unmodified $y(n) = u(n)$. The gradient of the cost function with respect to the parameter weights can then be calculated.

$$\begin{aligned} \nabla &= \frac{\partial J(\hat{\boldsymbol{h}}(n))}{\partial \hat{\boldsymbol{h}}^*(n)} \\ &= \frac{\partial |e(n)|^2}{\partial \hat{\boldsymbol{h}}^*(n)} \end{aligned} \tag{2.21}$$

For any complex number z we know that $|z|^2 = z(z^*)$; the above derivative can be simplified using this fact and the fact that $\hat{d}$ is not a function of $\hat{\boldsymbol{h}}(n)$.

$$\begin{aligned} \frac{\partial J(\hat{\boldsymbol{h}}(n))}{\partial \hat{\boldsymbol{h}}^*(n)} &= \frac{\partial \left(e(n)(e^*(n)\right)}{\partial \hat{\boldsymbol{h}}^*(n)} \\ &= e^*(n)\frac{\partial e(n)}{\partial \hat{\boldsymbol{h}}^*(n)} + e(n)\frac{\partial e^*(n)}{\partial \hat{\boldsymbol{h}}^*(n)} \\ &= e^*(n)\frac{\partial \left(\hat{d}(n) - \boldsymbol{y}^T(n)\hat{\boldsymbol{h}}(n)\right)}{\partial \hat{\boldsymbol{h}}^*(n)} + e(n)\frac{\partial \left(\hat{d}^*(n) - \boldsymbol{y}^H(n)\hat{\boldsymbol{h}}^*(n)\right)}{\partial \hat{\boldsymbol{h}}^*(n)} \\ &= -e(n)\boldsymbol{y}^H(n) = -e(n)\boldsymbol{y}^*(n) \end{aligned} \tag{2.22}$$

The transpose is removed in the last step so we get a column vector out, which we add

to the previous estimate. The update equation is then shown to be:

$$\begin{aligned}\hat{\boldsymbol{h}}(n+1) &= \hat{\boldsymbol{h}}(n) - \boldsymbol{\mu}\frac{\partial J(\hat{\boldsymbol{h}}(n))}{\partial \hat{\boldsymbol{h}}^*(n)} \\ &= \hat{\boldsymbol{h}}(n) + \boldsymbol{\mu} e(n)\boldsymbol{y}^*(n)\end{aligned} \tag{2.23}$$

The algorithm is run until the error goes below a set threshold before enabling the predistorter algorithm which is described next.

2.4.2 Predistorter Parameter Estimation

In the case of the predistorter, the implementation is more complicated as we are trying to reduce the error of the output of another nonlinear system; which is why a good PA estimation is important. In this case we define the error between the ideal linear gain output and the actual data from the power amplifier. The error is defined as $e(n) = d(n) - \hat{d}(n)$. Using a similar set up to the PA characterization, the cost function is the squared error.

$$J(\boldsymbol{w}(n)) = |e(n)|^2 = \left|d(n) - \hat{d}(n)\right|^2 \tag{2.24}$$

Using the definition for complex gradient descent:

$$\begin{aligned}\nabla &= \frac{\partial J(\boldsymbol{w}(n))}{\partial \boldsymbol{w}^*(n)} \\ &= \frac{\partial |e(n)|^2}{\partial \boldsymbol{w}^*(n)} \\ &= e^*(n)\frac{\partial e(n)}{\partial \boldsymbol{w}^*(n)} + e(n)\frac{\partial e^*(n)}{\partial \boldsymbol{w}^*(n)} \\ &= e^*(n)\frac{\partial\left(d(n) - \hat{d}(n)\right)}{\partial \boldsymbol{w}^*(n)} + e(n)\frac{\partial\left(d^*(n) - \hat{d}^*(n)\right)}{\partial \boldsymbol{w}^*(n)} \\ &= -e^*(n)\frac{\partial \hat{d}(n)}{\partial \boldsymbol{w}^*(n)} - e(n)\frac{\partial \hat{d}^*(n)}{\partial \boldsymbol{w}^*(n)}\end{aligned} \tag{2.25}$$

Looking at one of the parital derivatives in more detail, remembering that M is the PA memory size:

$$\frac{\partial \hat{d}(n)}{\partial \boldsymbol{w}^*(n)} = \sum_{r=0}^{M} \frac{\partial \hat{d}(n)}{\partial y^*(n-r)} \frac{\partial y^*(n-r)}{\partial \boldsymbol{w}^*(n)} \tag{2.26}$$

To simplify further, assume $w(n) \approx w(n-r)$ which leads to:

$$\frac{\partial y^*(n-r)}{\partial \boldsymbol{w}^*(n)} \approx \frac{\partial y^*(n-r)}{\partial \boldsymbol{w}^*(n-r)} \approx \boldsymbol{u}^*(n-r) \tag{2.27}$$

Following a similar process for the other partial derivative, the gradient can be rewritten.

$$\frac{\partial J(\boldsymbol{w}(n))}{\partial \boldsymbol{w}^*(n)} = -e^*(n) \sum_{r=0}^{M} \frac{\partial \hat{d}(n)}{\partial y^*(n-r)} \boldsymbol{u}^*(n-r) - e(n) \sum_{r=0}^{M} \frac{\partial \hat{d}^*(n)}{\partial y^*(n-r)} \boldsymbol{u}^*(n-r) \tag{2.28}$$

Looking at the first partial derivative, knowing it will be zero for any delay q not equal to r:

$$\begin{aligned}
\frac{\partial \hat{d}(n)}{\partial y^*(n-r)} &= \sum_{k=0}^{K} \hat{h}_{kr} \frac{\partial \left(y(n-r) |(y(n-r)|^k \right)}{\partial y^*(n-r)} \\
&= \sum_{k=0}^{K} \hat{h}_{kr} \frac{\partial \left(y(n-r) y(n-r)^{\frac{k}{2}} y^*(n-r)^{\frac{k}{2}} \right)}{\partial y^*(n-r)} \\
&= \sum_{k=0}^{K} \hat{h}_{kr} \frac{\partial \left(y(n-r)^{\frac{k+2}{2}} y^*(n-r)^{\frac{k}{2}} \right)}{\partial y^*(n-r)}
\end{aligned} \tag{2.29}$$

Using the definition of a derivative by a complex conjugate, the final result is:

$$\frac{\partial \hat{d}(n)}{\partial y^*(n-r)} = \sum_{k=0}^{K} k \hat{h}_{kr} |y(n-r)|^{k-2} y(n-r)^2 \tag{2.30}$$

Using the same process for the second partial derivative we get:

$$\begin{aligned}
\frac{\partial \hat{d}^*(n)}{\partial y^*(n-r)} &= \sum_{k=0}^{K} \hat{h}_{kr}^* \frac{\partial \left(y^*(n-r) |(y(n-r)|^k \right)}{\partial y^*(n-r)} \\
&= \sum_{k=0}^{K} \hat{h}_{kr}^* \frac{\partial \left(y^*(n-r) y(n-r)^{\frac{k}{2}} y^*(n-r)^{\frac{k}{2}} \right)}{\partial y^*(n-r)} \\
&= \sum_{k=0}^{K} \hat{h}_{kr}^* \frac{\partial \left(y^*(n-r)^{\frac{k+2}{2}} y(n-r)^{\frac{k}{2}} \right)}{\partial y^*(n-r)} \\
&= \sum_{k=0}^{K} (k+2) \hat{h}_{kr}^* |y(n-r)|^k
\end{aligned} \tag{2.31}$$

Putting it all together, the gradient is:

$$\begin{aligned}
\frac{\partial J(\boldsymbol{w}(n))}{\partial \boldsymbol{w}^*(n)} = &- e^*(n) \sum_{r=0}^{M} \left(\sum_{k=0}^{K} k \hat{h}_{kr} |y(n-r)|^{k-2} y(n-r)^2 \right) \boldsymbol{u}^*(n-r) \\
&- e(n) \sum_{r=0}^{M} \left(\sum_{k=0}^{K} (k+2) \hat{h}_{kr}^* |y(n-r)|^k \right) \boldsymbol{u}^*(n-r)
\end{aligned} \tag{2.32}$$

The update equation is then shown to be:

$$\begin{aligned}
\boldsymbol{w}(n+1) &= \boldsymbol{w}(n) - \boldsymbol{\mu} \frac{\partial J(\hat{\boldsymbol{h}}(n))}{\partial \hat{\boldsymbol{h}}^*(n)} \\
&= \boldsymbol{w}(n) + \boldsymbol{\mu} e^*(n) \sum_{r=0}^{M} \left(\sum_{k=0}^{K} k \hat{h}_{kr} |y(n-r)|^{k-2} y(n-r)^2 \right) \boldsymbol{u}^*(n-r) \\
&\quad + \boldsymbol{\mu} e(n) \sum_{r=0}^{M} \left(\sum_{k=0}^{K} (k+2) \hat{h}_{kr}^* |y(n-r)|^k \right) \boldsymbol{u}^*(n-r)
\end{aligned} \tag{2.33}$$

2.5 Algorithm Development and Testing

To ensure robust testing and development of the gradient descent algorithm, a MATLAB simulation was used initially to create a benchmark of expected results. This was then

converted into a Simulink block diagram to create the parallel structure that would be used in the FPGA. Finally, the diagram was converted into a fixed point VHDL implementation and simulated. All three steps included a memoryless PA and a nonlinear PA with memory effects to both characterize and linearize.

CHAPTER 3

RESULTS

The performance of the algorithms and implementations will be based on the Nominal Mean Squared Error (NMSE). The success of the PA characterization is the NMSE between the model output and the actual output of the PA. The full DPD system performance will be based on the improvement in NMSE between the desired linear output and the actual final PA output. The results are also visualized using Power Spectral Density (PSD) plots, a representation of a signals power content versus frequency.

A couple of PA models have been developed for testing purposes. One representing a device without memory effects using an inverse tangent function, and a Wiener-Hammerstein model for the more complex case. The results of the DPD and PA characterization are visualized using Power Spectral Density (PSD) plots, a representation of a signals power content versus frequency.

3.1 Memoryless and Wiener-Hammerstein PA Models

The memoryless PA model is taken from [29] and is represented by the following equation:

$$\hat{d}(n) = \left(\gamma_1 \tan^{-1}\left(\zeta_1|y(n)|\right) + \gamma_2 \tan^{-1}\left(\zeta_2|y(n)|\right)\right) \exp j\angle y(n) \tag{3.1}$$

Where $\gamma_1 = 8.00335 - j4.61157$, $\gamma_2 = -3.77167 + j12.03758$, $\zeta_1 = 2.26895$ and $\zeta_2 = 0.8234$. The ideal gain for this system is $G = 10$, an example of the distortion caused by the PA is shown in Figure 3.1.

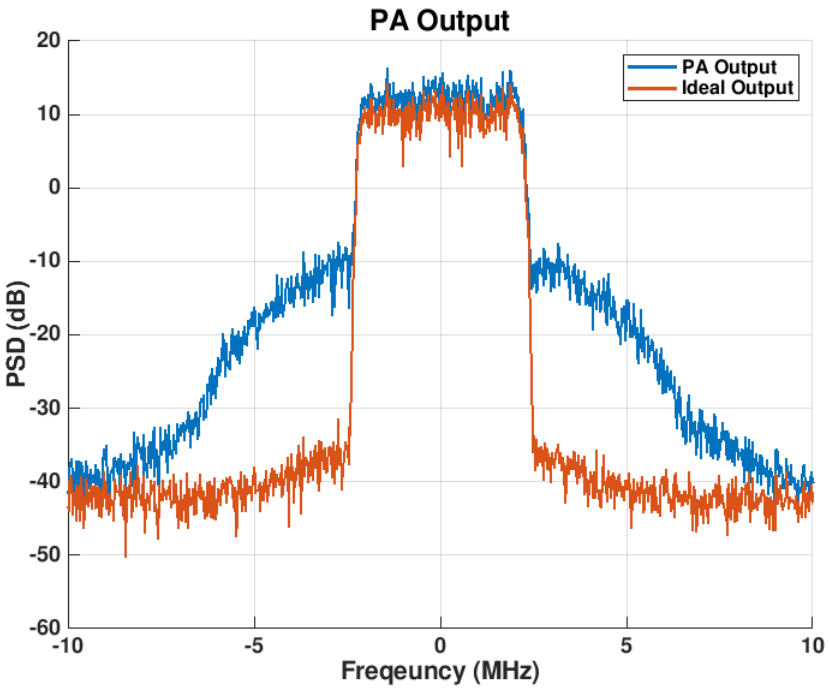

Figure 3.1: Arctangent Model

A Wiener-Hammerstein model is a mathematical model that contains a Linear Time-Invariant (LTI) system, followed by a memoryless nonlinearity, followed by another LTI system [34]. The following image shows the block structure of this type of model [35]:

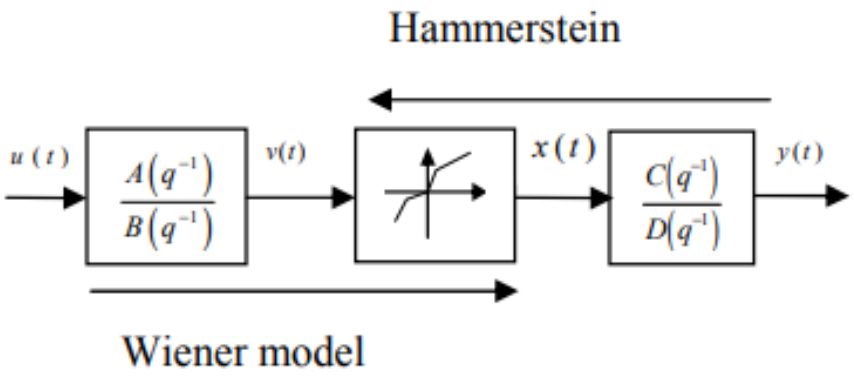

Figure 3.2: Wiener-Hammerstein Model

For testing, the nonlinear component is represented by the same inverse tangent func-

tion described above, the LTI systems before and after the nonlinear block are:

$$H(z) = \frac{1}{1.5}\frac{1+0.25z^{-2}}{1+0.4z^{-1}} \; or \; y(n) = \frac{1}{1.5}\left(x(n) + 0.25x(n-2) - 0.6y(n-1)\right) \quad (3.2)$$

$$G(z) = \frac{1}{0.52}\frac{1-0.1z^{-1}}{1-0.2z^{-1}} \; or \; y(n) = \frac{1}{0.52}\left(x(n) - 0.1x(n-1) + 0.104y(n-1)\right) \quad (3.3)$$

This system results in an ideal gain $G = 15$ with the distortion shown in Figure 3.3:

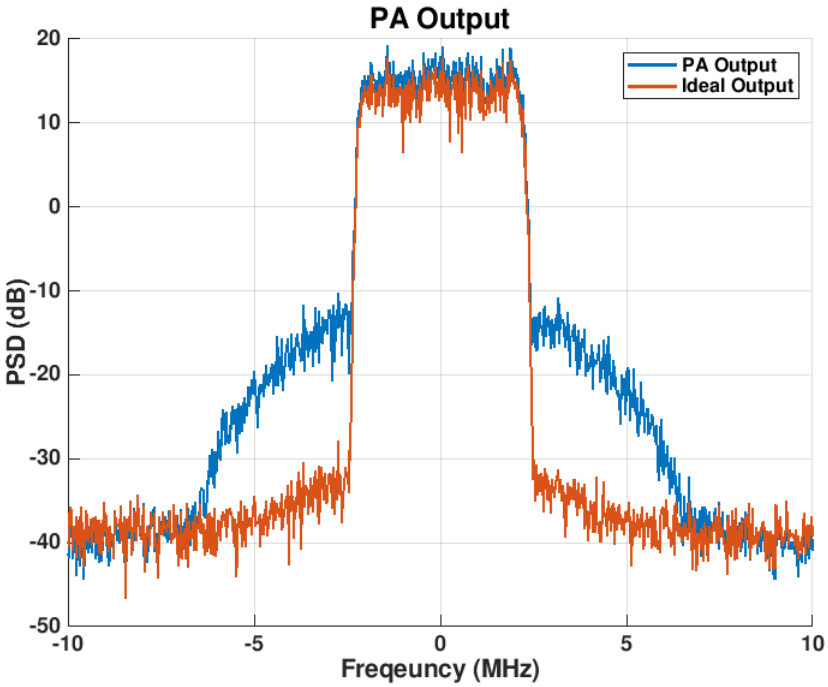

Figure 3.3: Wiener-Hammerstein PA Model Output

Although it may appear the Wiener-Hammerstein distortion is less intense than the arctangent model, it is important to remember due to the memory effects, predistortion is more complex in the Wiener-Hammerstein case.

3.2 MATLAB Implementation and Results

The first step in development was developing a working MATLAB model to verify the optimal performance of the developed algorithm. Only after this verification was completed, would it be worth-wile to begin Simulink and HDL development.

3.2.1 Memoryless PA

For the memoryless PA, a polynomial order of 13 was used, with no memory terms included. The polynomial order was kept the same for the PA characterization block and the DPD block for simplicity.

PA Characterization

Performing PA characterization of the PA using the memory polynomial model results in a NMSE of -47.0037 between the actual and model output, within 187,000 iterations of the gradient descent. The actual PA output with the output of the estimated PA model is shown in Figure 3.4 below, along with the convergence of the calculated cost function over time in Figure 3.5. It can be seen that the estimated output and model output are nearly identical.

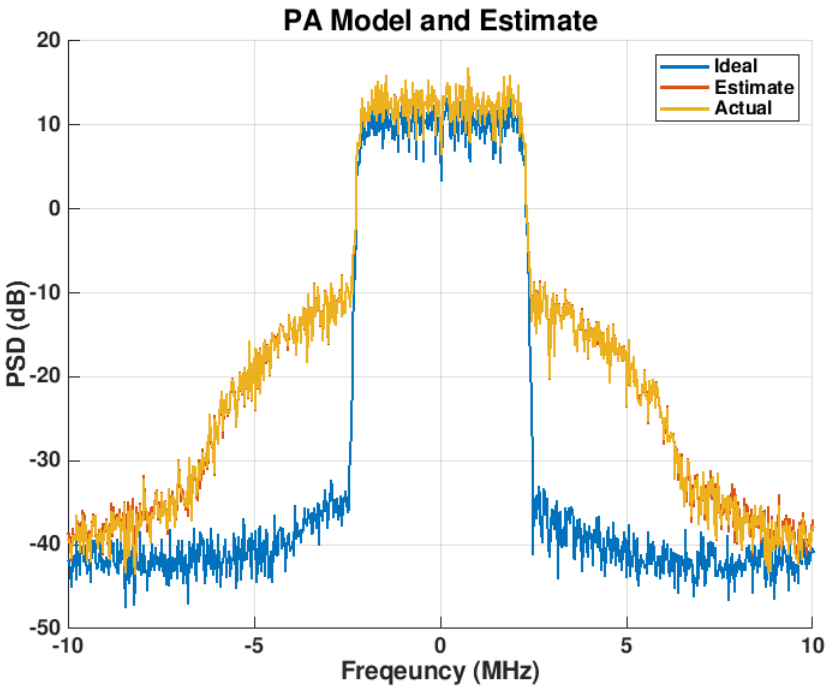

Figure 3.4: Arctan PA Output and Estimated Model Output

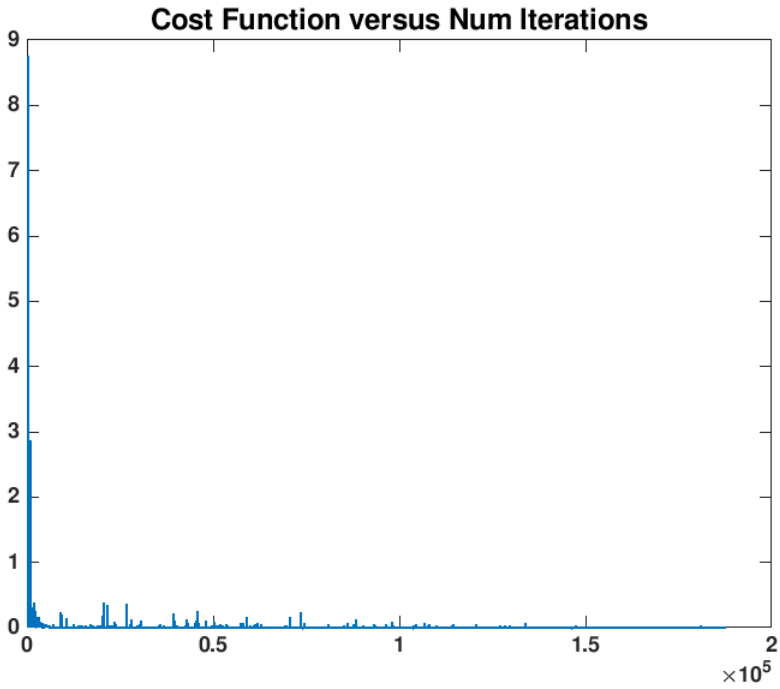

Figure 3.5: Convergence of the Cost Function Arctangent Model

In comparison, a Least-Squares solution results in an NMSE of -46.1848, with the same amount of data. This is extremely undesirable considering the worse performance requires large matrix inversions that would be impossible to perform on an FPGA. An orthogonalized basis Least-Squares approach does perform significantly better, resulting in an NMSE of -109.3782. However, its much larger computational complexity is hard to justify considering the loss in performance when memory effects are added. The feasibility of implementing the orthogonal basis in an HDL architecture is also very low.

Full Digital Predistortion

With no digital predistortion, the overall system has a NMSE of -10.46 between the ideal linear gain and the actual PA output. Adding the predistortion block with the gradient descent algorithm reduced the error to -41.759, resulting in a large decrease in the nonlinear behavior of the PA, as shown in Figure 3.6. Error convergence over time is also shown, with one sample being used per iteration.

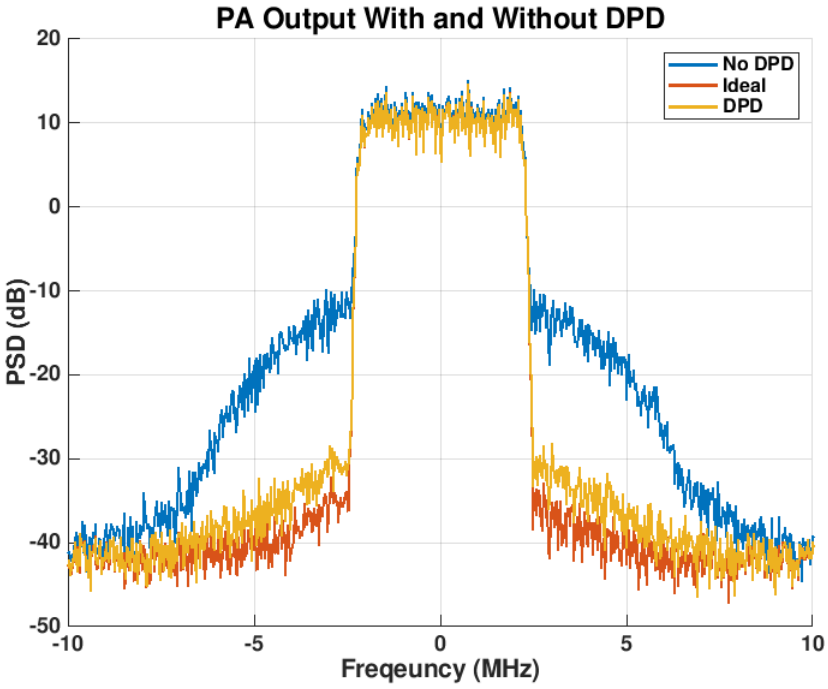

Figure 3.6: Arctan PA Output Using Predistortion Algorithm

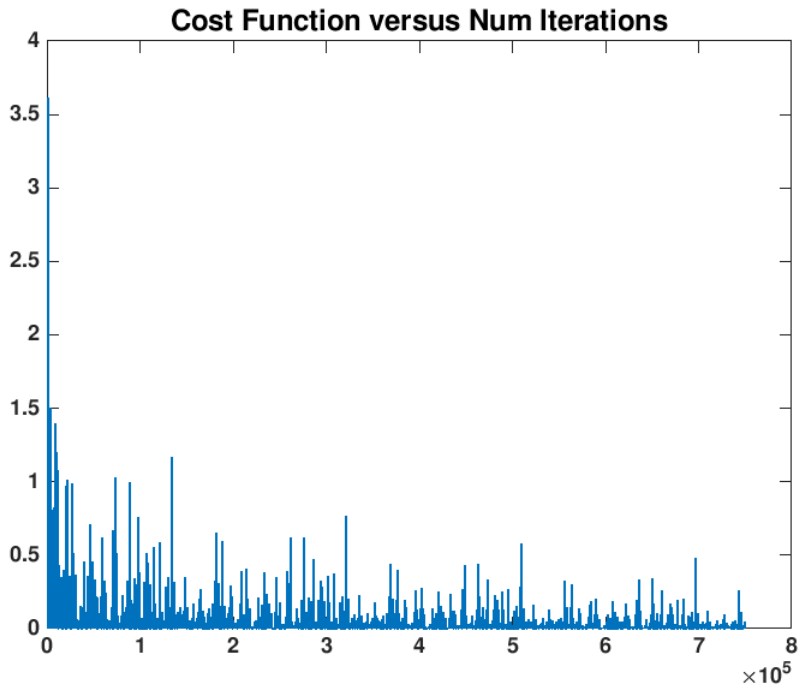

Figure 3.7: Convergence of the Cost Function Arctangent Predistortion

Using a Least-Squares indirect learning scheme and the same number of data points, the NMSE was driven down to -37.5813 which is worse when compared to the direct learning approach. The ill-conditioned matrix also makes this approach both more computationally expensive in terms of matrix inversion, but also numerically unstable. Using the orthogonalized basis for the indirect learning approach produces an NMSE of -41.5654, still slightly lower than the developed gradient descent approach.

3.2.2 Wiener-Hammerstein PA

For the Wiener-Hammerstein PA, a polynomial order of 13 was used, with 3 memory terms included. The polynomial order was kept the same for the PA characterization block and the DPD block for simplicity.

PA Characterization

Performing PA characterization of the PA using the memory polynomial model results in a NMSE of -45.6153 between the actual and model output, within 187,000 iterations of the gradient descent. The actual PA output with the output of the estimated PA model is shown in Figure 3.8 below, along with the convergence of the calculated cost function over time in Figure 3.9. It can be seen that the estimated output and model output are nearly identical and that the cost function converges to a minimum extremely quickly.

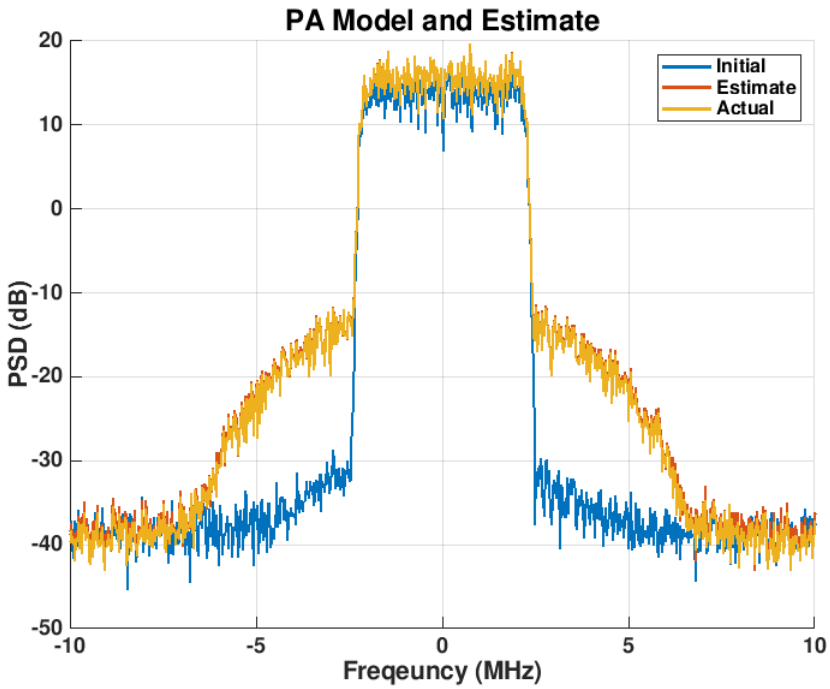

Figure 3.8: Wiener-Hammerstein PA Output and Estimated Model Output

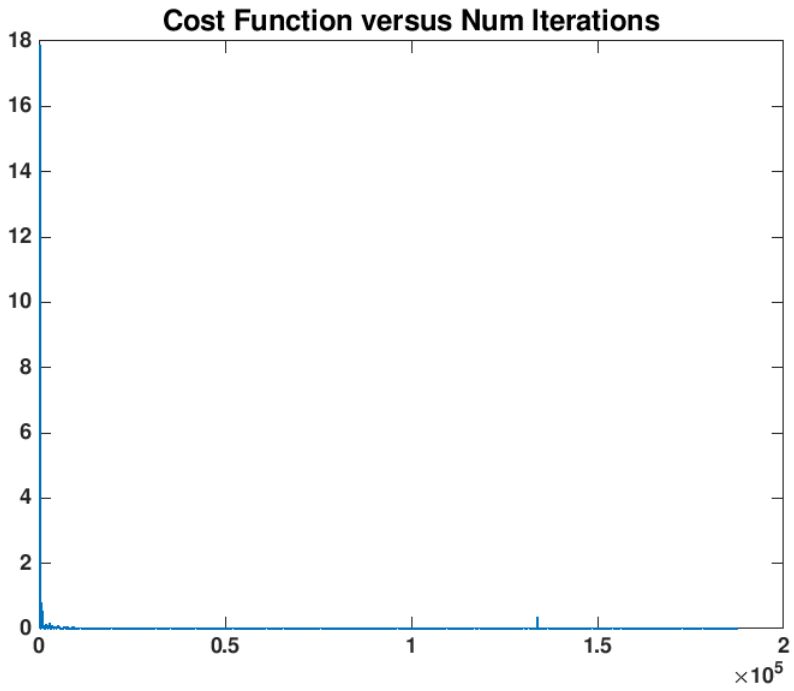

Figure 3.9: Convergence of the Cost Function Wiener-Hammerstein Model

In comparison, a Least-Squares solution results in an NMSE of -7.7910, with the same amount of data. This performance also falls victim to ill-conditioned matrices is below the gradient descent benchmark, making the algorithm a poor choice. An orthogonalized basis Least-Squares approach does perform better, resulting in an NMSE of -54.1879, but this is a large fall in performance from the memoryless case, demonstrating the robustness of the gradient descent approach.

Full Digital Predistortion

With no digital predistortion, the overall system has a NMSE of -13.806 between the ideal linear gain and the actual PA output. Adding the predistortion block with the gradient descent algorithm reduced the error to -43.1815, resulting in a large decrease in the nonlinear behavior of the PA, as shown in Figure 3.10. Error convergence over time is also shown, with one sample being used per iteration.

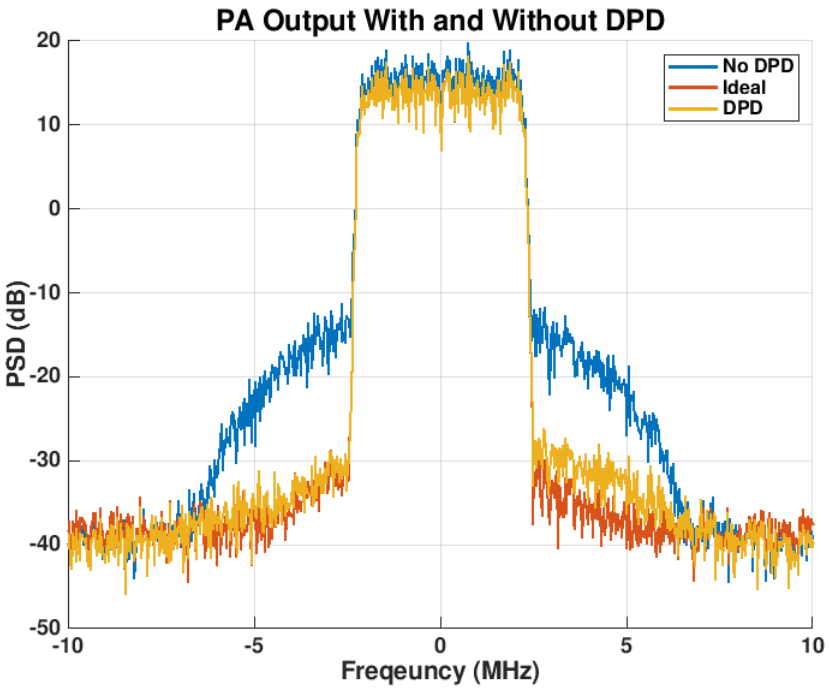

Figure 3.10: Wiener-Hammerstein PA Output Using Predistortion Algorithm

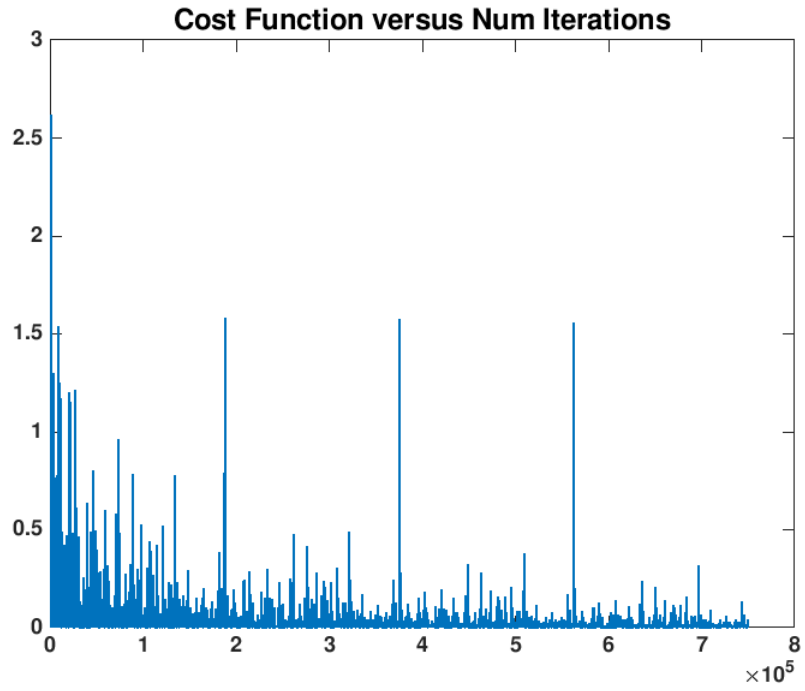

Figure 3.11: Convergence of the Cost Function Wiener-Hammerstein Predistortion

Figure 3.11 shows that some sample outliers do not have a large effect on the overall performance, some deviations do occur during the sample by sample descent approach but the cost function still converges over time.

Using a Least-Squares indirect learning scheme and the same number of data points, the NMSE was driven down to -5.9964 which is worse than having no predistortion at all, demonstrating the drawbacks of indirect learning. Using the orthogonalized basis for the indirect learning approach produces an NMSE of -10.8660, still slightly lower than having no predistortion implemented.

The MATLAB simulation is able to show robust performance of the gradient descent algorithm compared to other common approaches to digital predistortion as well as verify the mathematical derviation for the complex gradient descent. The implementation performs well for PA characterization and full DPD in both memryless systems and devices with present memory effects. Thus the implementation provides a balance between robustness and performance over multiple types of nonlinear behavior while still being computationally feasible in hardware.

3.3 Simulink Implementation and Results

Converting the MATLAB model to Simulink is an important step in developing HDL as it allows the development of the block diagram structure to be used in the final implementation. In order to conform to HDL design and make the translation process easier, traditional HDL blocks such as Look Up Tables (LUTs) and First In First Out (FIFO) data structures were used.

Look Up Tables are memory arrays used for function approximation via indexing. The input to the function is used as the index to the array and the value stored at that index is the desired output value of the function. LUTs have a trade off between size and accuracy, smaller tables take up less memory but are unable to store as many indices, which could result in poor performance for missing values. Various interpolation schemes have been

developed to manage this balance and LUTs are widely used throughout DPD implementations [26]. In this implementation LUTs are used to approximate the memory polynomial orders, $|y(n-r)|^k$, thus preventing the need for large multiplication trees. Although no interpolation schemes were used in this initial architecture, that is something that can be improved upon in future iterations.

The PA coefficients are stored using registers, when a new sample is received, the current coefficient $\hat{\boldsymbol{h}}(n)$ is used for the PA output estimation. Once the gradient and updated coefficients are calculated, the updated values, $\hat{\boldsymbol{h}}(n+1)$, are stored into their respective registers and the process repeats.

3.3.1 Memoryless PA

For consistent comparison to the other implementations, a polynomial order of 13 was used, with no memory terms included. The polynomial order was kept the same for the PA characterization block and the DPD block.

PA Characterization

For reference, the PA model memory polynomial without memory is written as:

$$\hat{d}(n) = \sum_{p=0}^{P} h_p y(n)|y(n)|^p \tag{3.4}$$

The block structure of the PA characterization is shown in Figure 3.12. The various powers of the magnitude of the input are calculated based on LUTs using the input sample's magnitude square as the index. This creates a bus signal that is sent to the estimation block which stores the estimated coefficients.

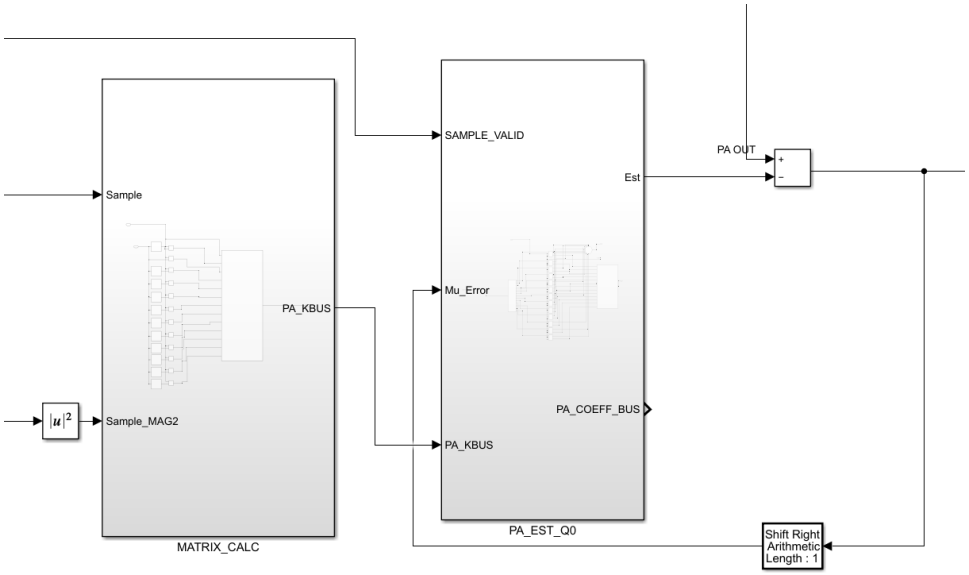

Figure 3.12: Simulink Block Structure ATAN PA

Verification of the Simulink model was done via the same NMSE metric as well as visual confirmation of the convergence of the cost function. Figure 3.13 shows how well the estimate matches the actual PA output.

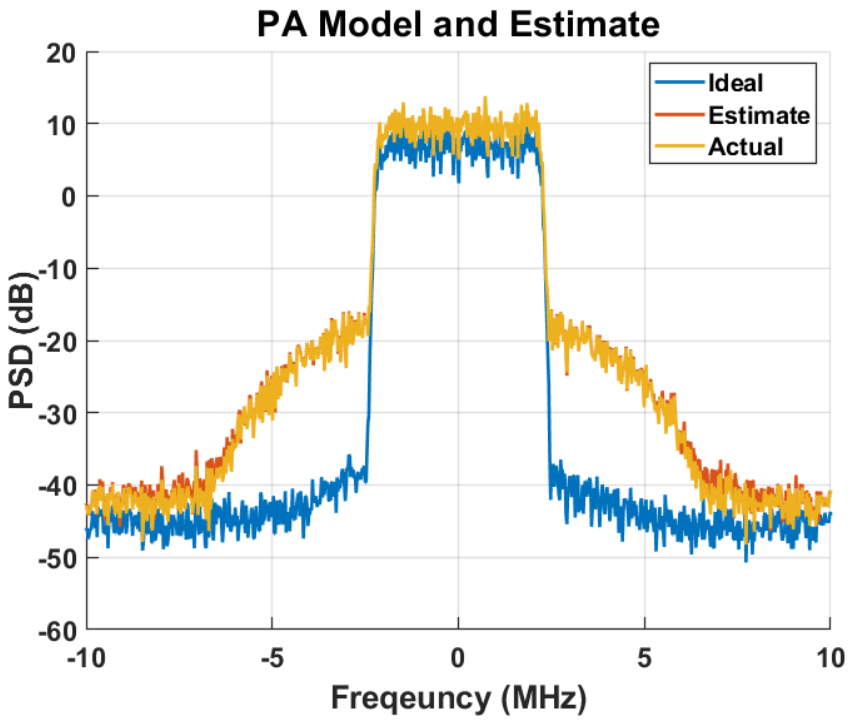

Figure 3.13: Simulink Arctan PA Output and Estimated Model Output

In order to measure the NMSE between the estimated PA model and the actual output, the coefficients were exported to MATLAB and the calculation was done. Unsurprisingly, the normalized error was measured to be -46.117 which is nearly identical to the performance obtained from the MATLAB model. Slight deviations were to be expected due to approximation errors of the LUTs but this serves as a sanity check that the block system meets the benchmark set by the model.

Full Digital Predistortion

The full DPD system is more complex, due to the calculation of the gradient for the predistorter coefficients. Since both PA characterization and predistortion are happening simultaneously, a state machine was designed to control the entire system. The diagram is shown in Figure 3.14.

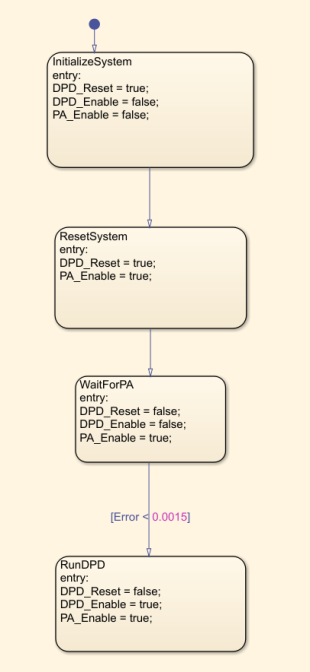

Figure 3.14: Digital Predistortion State Machine

The system is first initialized and reset, the coefficients of the predistorter are set such that the input signal passes through unmodified. The PA characterization block is then enabled, the output error is monitored by the state machine and once it falls below a predetermined threshold, the predistorter coefficient estimation is enabled. A moving average of size 256 is used on the PA output error to prevent any outliers causing the predistorter gradient descent to be enabled too early.

The update equation with no memory effect, for the DPD coefficients was calculated to be from (2.33):

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{\mu} e^*(n) \left(\sum_{k=0}^{K} k \hat{h}_k |y(n)|^{k-2} y(n)^2 \right) \boldsymbol{u}^*(n) + \boldsymbol{\mu} e(n) \left(\sum_{k=0}^{K} (k+2) \hat{h}_k^* |y(n)|^k \right) \boldsymbol{u}^*(n)$$

This can be rewritten as shown below, showing there is some overlap in computation of

$(k+2)|y(n)|^k$ between the two summations.

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{\mu} e^*(n) \left(\sum_{k=-2}^{K-2} (k+2) \hat{h}_{k-2} |y(n)|^k y(n)^2 \right) \boldsymbol{u}^*(n) + \boldsymbol{\mu} e(n) \left(\sum_{k=0}^{K} (k+2) \hat{h}_k^* |y(n)|^k \right) \boldsymbol{u}^*(n) \tag{3.5}$$

LUTs are again used for approximating the various powers of absolute values. The block diagram shown in Figure 3.15 shows that the two summations are split to take advantage of concurrent computation.

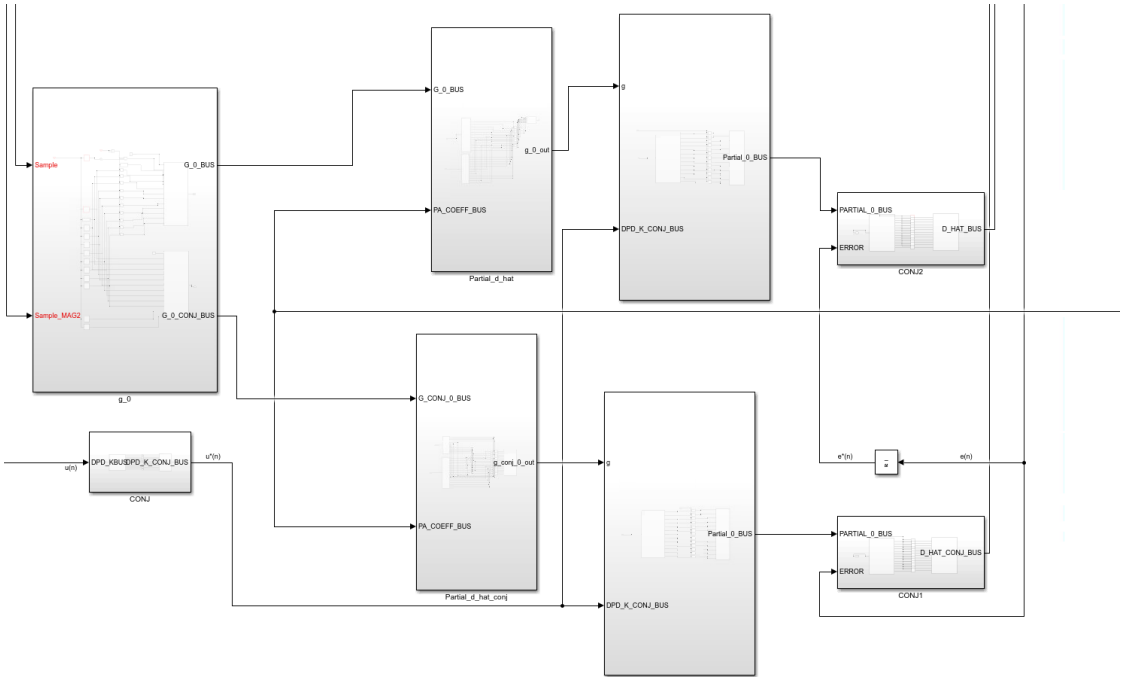

Figure 3.15: Digital Predistortion Block Diagram

G_0_Bus holds the values of $k|y(n)|^{k-2}y(n)^2$ and G_0_Conj_Bus represents $(k+2)|y(n)|^k$. These values are multiplied by the appropriate PA coefficient value $\hat{h}_k$ from the PA characterization block and summed. The calculated values of $\boldsymbol{u}^*(n)$ and $\boldsymbol{\mu}e(n)$ are then used to give the final results for the partial derivatives.

After simulation, the Simulink model produces a NMSE of -43.081 between the ideal

linear behavior and the overall DPD system. This value is actually larger than the NMSE of -41.759 produced using the MATLAB model. This is most likely due to the constant adaptive nature of the PA characterization that is running during the DPD calculation.

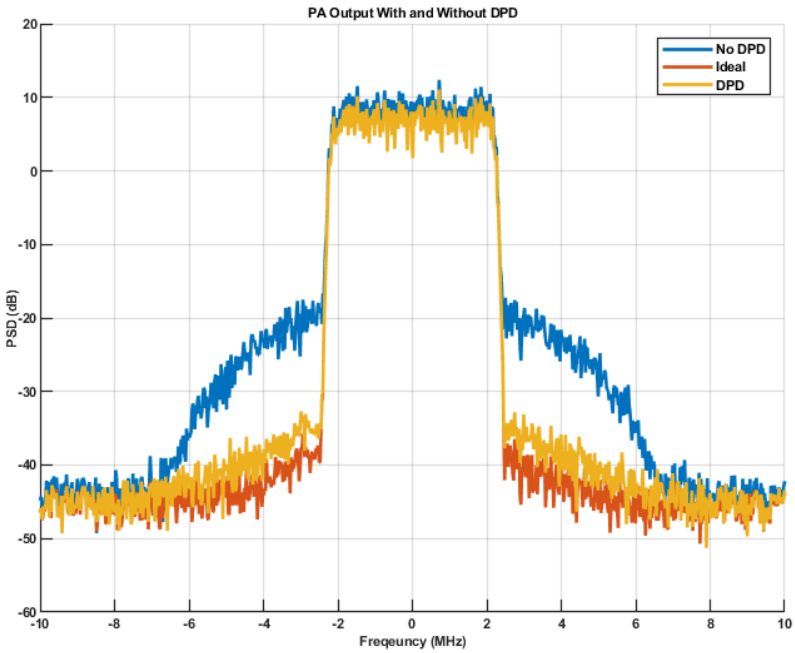

Figure 3.16: Simulink Arctan PA Output Using Predistortion Algorithm

3.3.2 Wiener-Hammerstein PA

For consistent comparison to the other implementations, a polynomial order of 13 was used, with 3 memory terms included. The polynomial order was kept the same for the PA characterization block and DPD.

PA Characterization

The gradient function for PA characterization is written as from (2.22):

$$\frac{\partial J(\hat{\boldsymbol{h}}(n))}{\partial \hat{\boldsymbol{h}}^{*}(n)} = -e(n)\boldsymbol{y}^{*}(n)$$

Where $\boldsymbol{y}(n)$ is a vector that can be expanded out as:

$$\boldsymbol{y}(n) = [\phi_{00}[y(n)] \;\; \phi_{10}[y(n)] \;\; ... \;\; \phi_{P1}[y(n)] \;\; ... \;\; \phi_{PM}[y(n)]]^T \tag{3.6}$$

Where $\phi_{pm}[y(n)] = y(n-m)|y(n-m)|^p$. Adding memory effects to the existing PA characterization structure is fairly straightforward. Since the memory polynomial calculation occurs for each sample, using registers and delay blocks these values are now available at later sample points. The simulation produces a PSD as shown in Figure 3.17 and has a NMSE of -46.524.

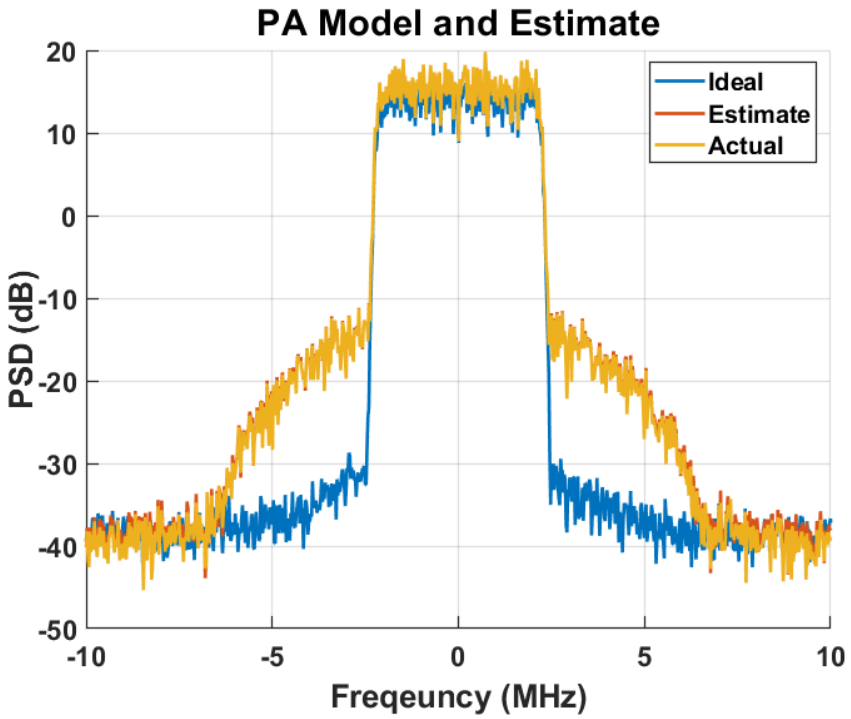

Figure 3.17: Simulink Wiener-Hammerstein PA Output and Estimated Model Output

Full Digital Predistortion

The predistortion gradient function is a little more complex due to the presence of summations and delay terms. From (2.33):

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{\mu} e^*(n) \sum_{r=0}^{M} \left(\sum_{k=0}^{K} k \hat{h}_{kr} |y(n-r)|^{k-2} y(n-r)^2 \right) \boldsymbol{u}^*(n-r)$$
$$+ \boldsymbol{\mu} e(n) \sum_{r=0}^{M} \left(\sum_{k=0}^{K} (k+2) \hat{h}^*_{kr} |y(n-r)|^k \right) \boldsymbol{u}^*(n-r)$$

Using the same approach as PA characterization, delay blocks and intermediate registers can be used to store values of $\boldsymbol{u}^*(n-r)$ during computation. Ensuring the right coefficients from the PA, $\hat{h}_{kr}$, are multiplied by the correct values is also important. The updated architecture produces the results shown in Figure 3.18.

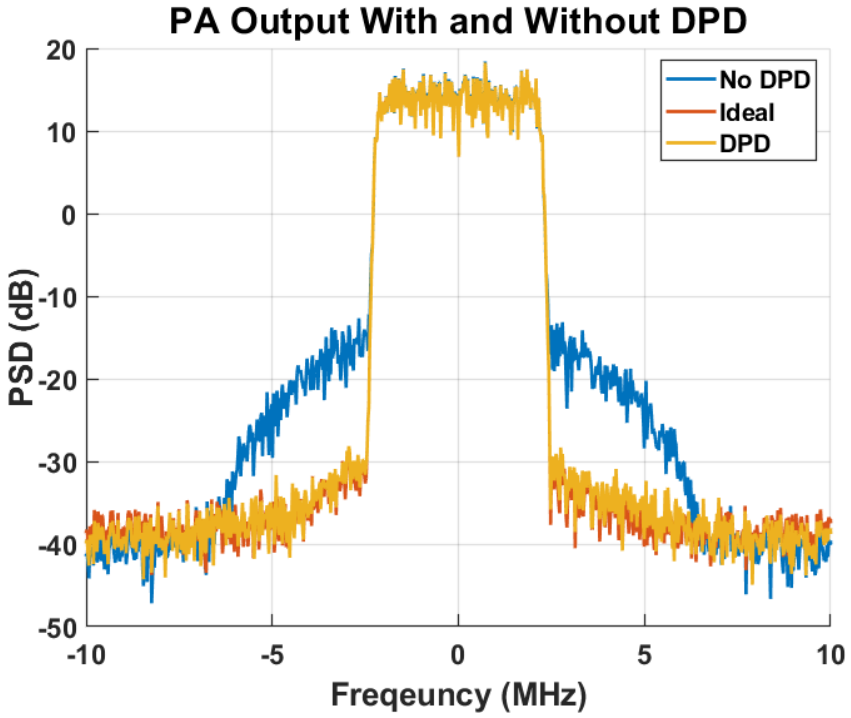

Figure 3.18: Simulink Wiener-Hammerstein PA Output Using Predistortion Algorithm

The algorithm results in a NMSE of -46.188 between the ideal linear gain and the output of the DPD system, around the same value obtained from the MATLAB model. However the Simulink structure was able to achieve this performance with approximately 25% of the input signal data. As with the memoryless model, this improved performance is likely

due to the constant adaptive nature of the PA model that was not present in the MATLAB model. This allows continued performance improvement as long as the system continues to run. This also demonstrates that the reduces performance in the PA characterization does not imply worse performance of the DPD system if they are run together.

The Simulink results show that the MATLAB performance can be replicated using a block diagram structure and LUTs. This makes the conversion into HDL much simpler as the architecture is mostly finalized at this point. This testing continues to show promising results that will translate well into a hardware implementation.

3.4 VHDL Implementation and Results

VHDL testing and simulation was done using the XilinX Vivado Design Suite with a test-bench written in SystemVerilog. The main challenge in converting the model to a pure HDL design was the introduction of fixed point arithmetic. MATLAB and Simulink both use double precision representations for any number, however, in an FPGA floating point numbers are much more difficult to implement. Floating point math is less efficient and uses a significant number of clock delays to compute. As a result, fixed point representation is used, despite its lower relative accuracy. Accounting for bit growth during operations is integral, as storing fewer bits is more memory efficient at the cost of accuracy.

The model was simulated with a 100 MHz clock and incoming data samples 50 MHz for simplicity. The simulation results of the HDL models are discussed below.

3.4.1 Memoryless PA

For consistent comparison to the other implementations, a polynomial order of 13 was used, with no memory terms included. The polynomial order was kept the same for the PA characterization block and the DPD block. The block diagram structure for the various implementation is kept constant from the Simulink simulations.

To measure the performance of the model, the converged polynomial coefficients were

exported into MATLAB and used to calculate the NMSE based on the given input data.

PA Characterization

Performing PA characterization on the memoryless PA model resulted in an NMSE of -46.4358 between the actual data output and the memory polynomial output. This is lower than the MATLAB model result of -47.0037 but this is to be expected due to losses in accuracy due to fixed-point representation and the use of LUTs. The generated PSD is shown in Figure 3.19 where it can be seen visually how well the estimate matches the actual data.

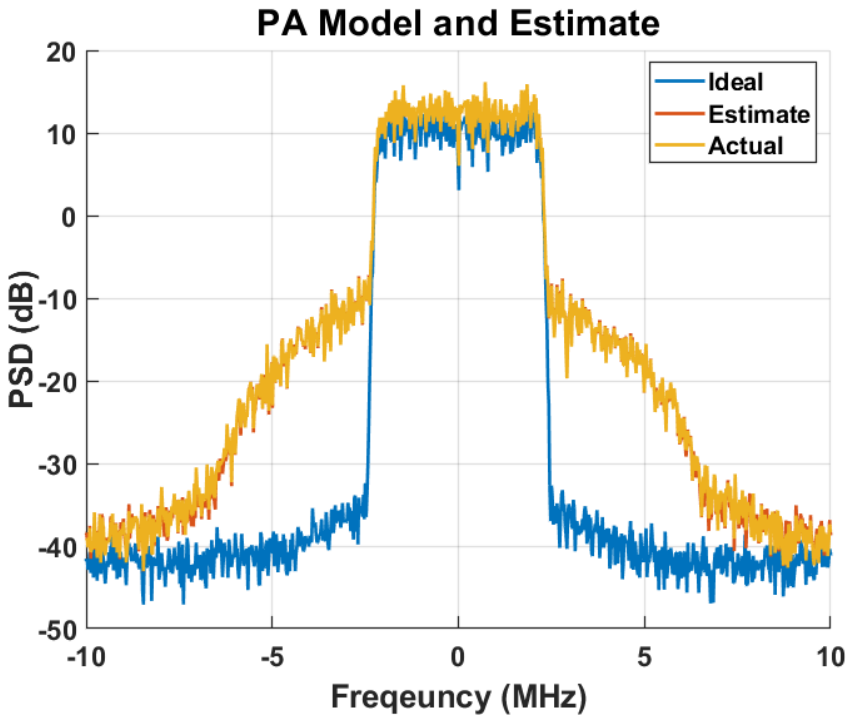

Figure 3.19: VHDL Arctan PA Output and Estimated Model Output

The simulation was run for a total of 2.4 ms with the NMSE converging as shown in the Figure below.

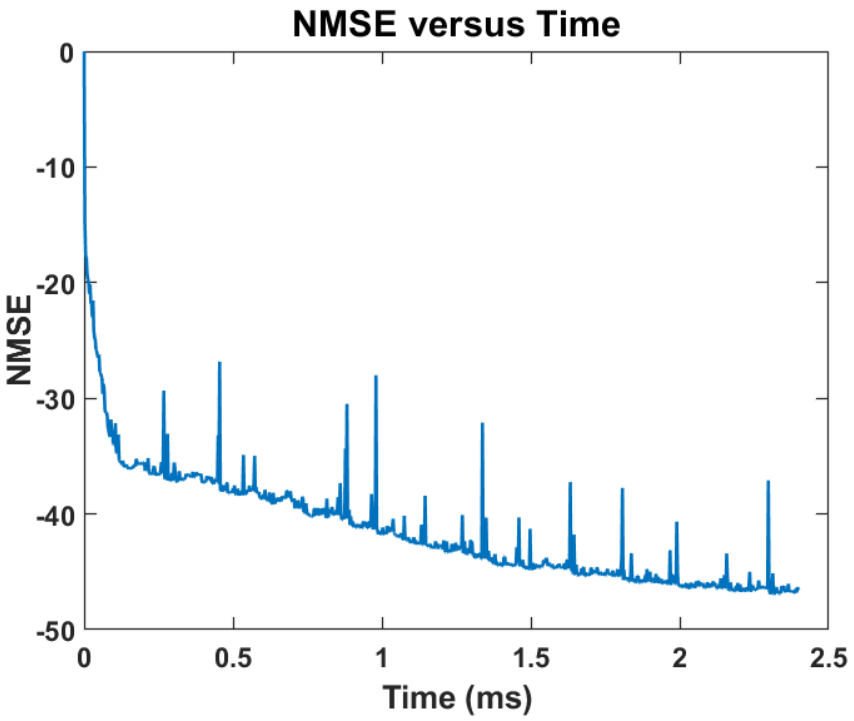

Figure 3.20: VHDL Arctan PA NMSE Versus Time

Full Digital Predistortion

For the memoryless predistortion system, the DPD gradient block was enabled after 10 μs indicating that the PA characterization had converged. The overall system was run for 9.6 ms before the NMSE was measured to be -41.5083 and the plot in Figure 3.21 was generated.

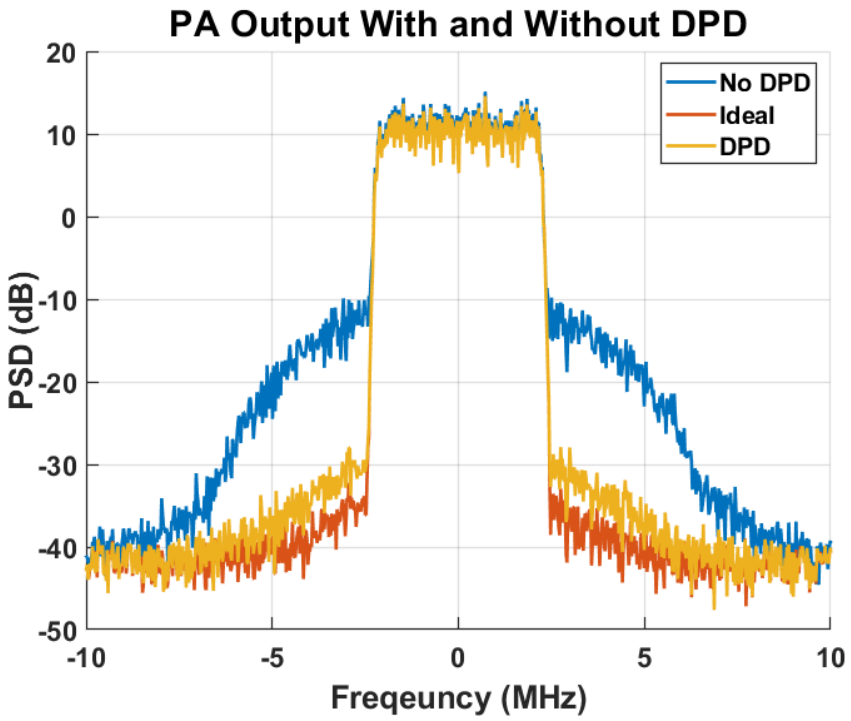

Figure 3.21: VHDL Arctan PA Output Using Predistortion Algorithm

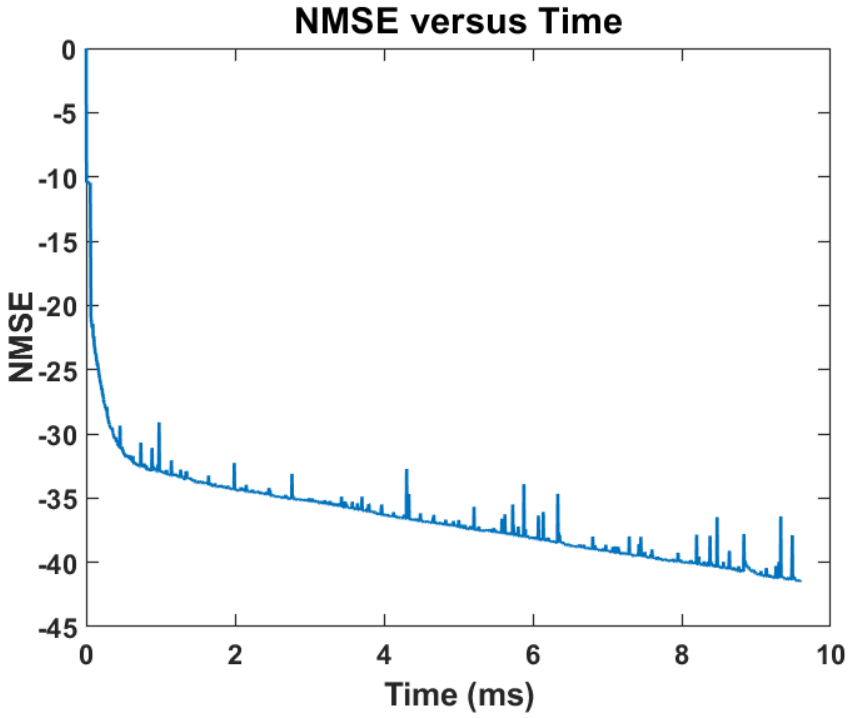

Figure 3.22: VHDL Arctan DPD NMSE Versus Time

3.4.2 Wiener-Hammerstein PA

For consistent comparison to the other implementations, a polynomial order of 13 was used, with 3 memory terms included. The polynomial order was kept the same for the PA characterization block and DPD. The block diagram structure for the various implementation is kept constant from the Simulink simulations.

To measure the performance of the model, the converged polynomial coefficients were exported into MATLAB and used to calculate the NMSE based on the given input data.

PA Characterization

Using memory terms, the PA was characterized with an NMSE of -42.62, resulting in the model shown in Figure 3.23. This is lower than the MATLAB simulation error of -45.6153, but this is once again likely due to fixed-point rounding errors and the LUT accuracy limitations.

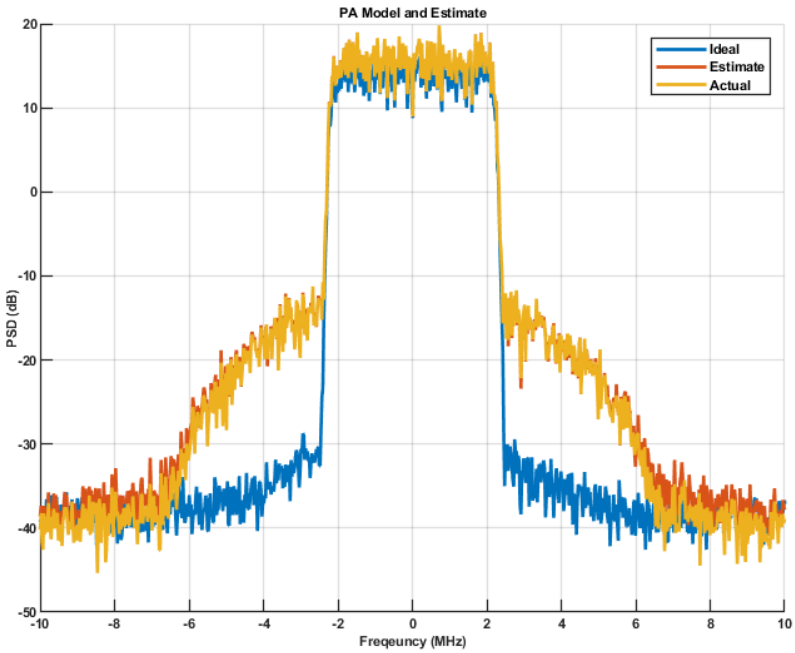

Figure 3.23: VHDL Wiener-Hammerstein PA Output and Estimated Model Output

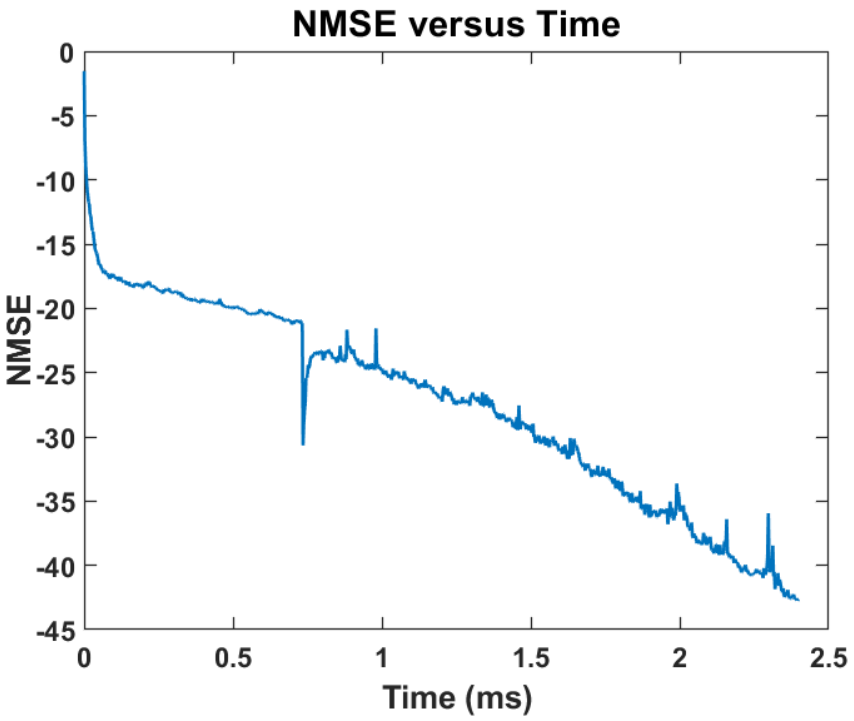

Figure 3.24: VHDL Wiener-Hammerstein PA NMSE Versus Time

Full Digital Predistortion

The digital predistorter performed more closely to the Simulink model with a NMSE of -45.657 between the ideal linear behavior and the actual DPD output. The results are shown in the following figures.

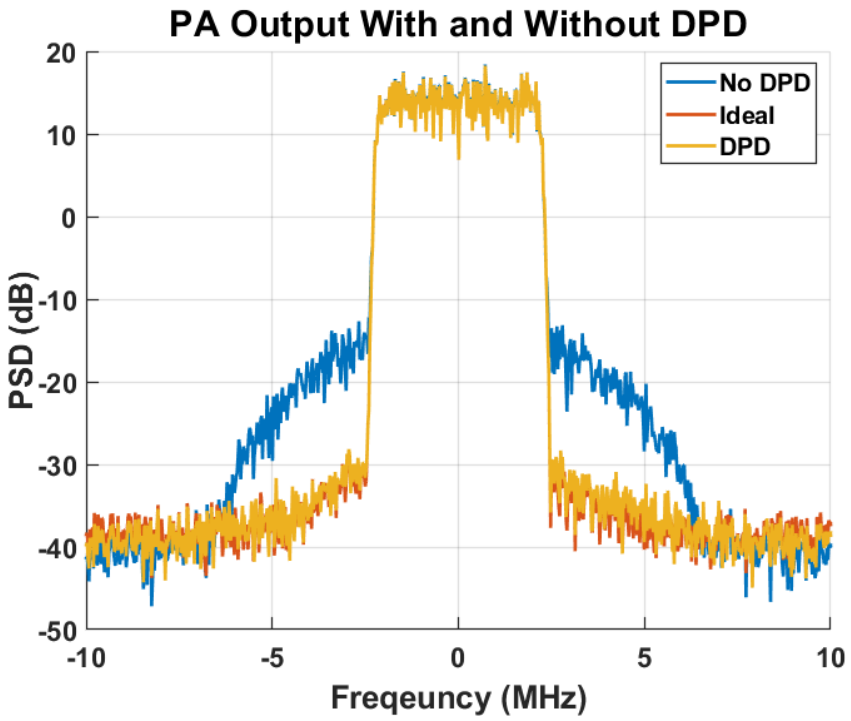

Figure 3.25: VHDL Wiener-Hammerstein PA Output Using Predistortion Algorithm

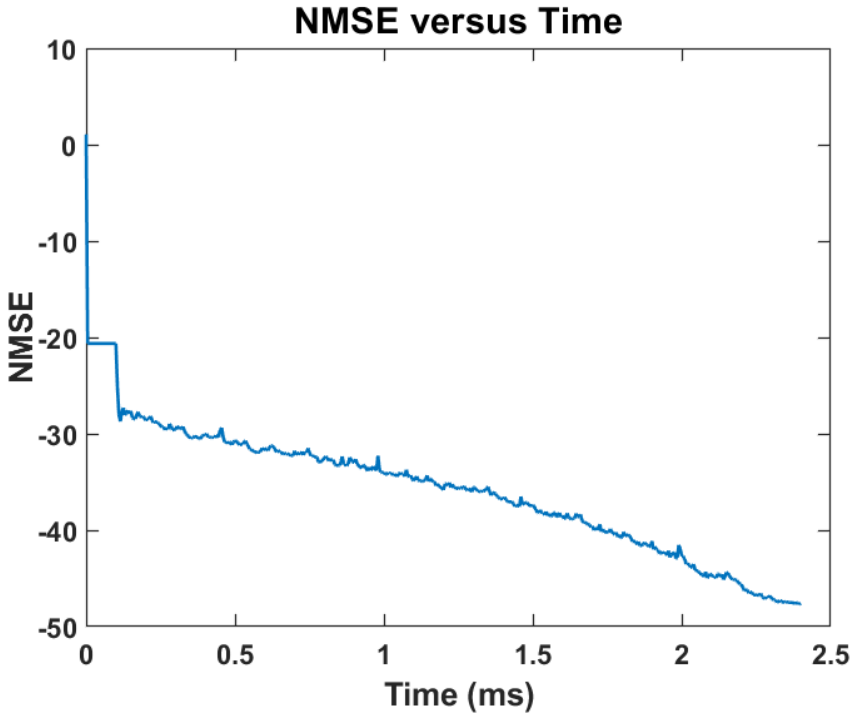

Figure 3.26: VHDL Wiener-Hammerstein DPD NMSE Versus Time

The results show constant performance improvement the longer the system is run, with an NMSE of around -60 being reached in less than 10 ms. For comparison, a matrix inversion in software can take hundreds of milliseconds to compute, not to mention the added computational complexity required to compute the orthogonal basis to ensure a well-conditioned matrix.

It is also important to note that varying clock and sample speeds will affect the convergence rate of the algorithm.

CHAPTER 4
DISCUSSION

The results of the research developed a robust Digital Predistortion scheme that utilizes a complex gradient descent approach for a fully hardware implementation. The performance was maintained throughout testing and development from a software model to a VHDL simulation. Low NMSE values and high conversion rates show the goals of the research were met. While some hardware implementations exist, the proposed solution uses the more robust direct learning architecture and gradient descent, removing the risks of numerical instability that are present in indirect learning solutions and least-squares implementations. The implementation outperformed these other approaches in systems with and without memory effects.

If the PA devices were to be characterized prior to deployment, convergence speed would be even shorter as the initial conditions would be close to the ideal values to begin with. The adaptive nature of the system would still be necessary to match any parameter drifts that the PA undergoes, however this process would be a lot faster than characterizing the PA and predistortion system from scratch.

4.1 Future Work

The research demonstrated the potential effectiveness and robustness of an HDL implementation of digital predistortion. However, there are still steps that can be taken to improve the performance and efficiency of the overall system. The current LUT method is highly memory inefficient, as every binary value for the magnitude square must have its own index, making the tables incredibly large. One improvement would be to utilize interpolation methods to reduce the size of the LUTs without any large sacrifices in accuracy. This would reduce the memory space on the FPGA at a higher computational cost. Two-dimensional

LUTs could also be implemented as it could simplify some of the complex multiplies that are being used. Both of these suggestions would demand further research and testing to find the most optimal choice or combination of choices for the specific implementation platform.

Further research into data path requirements can be done, especially with sub-sampling implementations becoming more common [19]. This would reduce the bandwidth requirements for the feedback path, resulting in a more energy efficient system.

Another potential improvement would be the ability to make the nonlinear order and memory depth as parameters that could be set in real time. This would allow for more flexible performance, ensuring higher efficiency for less complex power amplifiers. However, since the design is a hardware implementation, there would be no way to add or remove blocks during run time. This would mean that the implementation would take up the same amount of space on the fabric, regardless of the parameter values, but the run time power consumption could be improved.

Other improvements may only be seen once the development reaches physical hardware, but the thesis demonstrates that this implementation is a step in the right direction.

REFERENCES

[1] L. Ding, G. T. Zhou, D. R. Morgan, Z. Ma, J. S. Kenney, J. Kim, and C. R. Giardina, "A robust digital baseband predistorter constructed using memory polynomials," *IEEE Transactions on communications*, vol. 52, no. 1, pp. 159–165, 2004.

[2] J. Wood, *Behavioral Modeling and Linearization of RF Power Amplifiers*. Artech House, 2014.

[3] *Theory of operation for digital predistortion*, 2008.

[4] N. Binu and C. Suriyakala, "A weighted ofdm scheme with hadamard transform and windowing techniques for papr reduction," in *2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, IEEE, 2015, pp. 565–570.

[5] M. Senst and G. Ascheid, "Optimal output back-off in ofdm systems with nonlinear power amplifiers," in *2009 IEEE International Conference on Communications*, IEEE, 2009, pp. 1–6.

[6] D. Zhou and V. E. DeBrunner, "Novel adaptive nonlinear predistorters based on the direct learning algorithm," *IEEE transactions on signal processing*, vol. 55, no. 1, pp. 120–133, 2007.

[7] J. H. Vuolevi, T. Rahkonen, and J. P. Manninen, "Measurement technique for characterizing memory effects in rf power amplifiers," *IEEE Transactions on Microwave Theory and Techniques*, vol. 49, no. 8, pp. 1383–1389, 2001.

[8] Y. Wan, T. J. Dodd, and R. F. Harrison, "Modeling of power amplifier nonlinearities using volterra series," in *Proc. of the 16th IFAC Congress on Automatic Control*, 2005, pp. 785–790.

[9] A. Zhu, J. C. Pedro, and T. J. Brazil, "Dynamic deviation reduction-based volterra behavioral modeling of rf power amplifiers," *IEEE Transactions on microwave theory and techniques*, vol. 54, no. 12, pp. 4323–4332, 2006.

[10] J. Kim and K Konstantinou, "Digital predistortion of wideband signals based on power amplifier model with memory," *Electronics Letters*, vol. 37, no. 23, pp. 1417–1418, 2001.

[11] M. A. Hussein, V. A. Bohara, and O. Venard, "On the system level convergence of ila and dla for digital predistortion," in *2012 International Symposium on Wireless Communication Systems (ISWCS)*, IEEE, 2012, pp. 870–874.

[12] H. Paaso and A. Mammela, "Comparison of direct learning and indirect learning predistortion architectures," in *Wireless Communication Systems. 2008. ISWCS'08. IEEE International Symposium on*, IEEE, 2008, pp. 309–313.

[13] Y. Beltagy, P. Mitran, and S. Boumaiza, "Direct learning algorithm for digital predistortion training using sub-nyquist intermediate frequency feedback signal," *IEEE Transactions on Microwave Theory and Techniques*, vol. 67, no. 1, pp. 267–277, 2018.

[14] Z. Wang, W. Chen, G. Su, F. M. Ghannouchi, Z. Feng, and Y. Liu, "Low computational complexity digital predistortion based on direct learning with covariance matrix," *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, no. 11, pp. 4274–4284, 2017.

[15] L. Guan and A. Zhu, "Low-cost fpga implementation of volterra series-based digital predistorter for rf power amplifiers," *IEEE Transactions on Microwave Theory and Techniques*, vol. 58, no. 4, pp. 866–872, 2010.

[16] P. L. Gilabert, A. Cesari, G. Montoro, E. Bertran, and J.-M. Dilhac, "Multi-lookup table fpga implementation of an adaptive digital predistorter for linearizing rf power amplifiers with memory effects," *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, no. 2, pp. 372–384, 2008.

[17] H. Huang, P. Mitran, and S. Boumaiza, "Digital predistortion function synthesis using undersampled feedback signal," *IEEE Microwave and Wireless Components Letters*, vol. 26, no. 10, pp. 855–857, 2016.

[18] *Zynq ultrascale rfsoc*.

[19] N. Guan, N. Wu, and H. Wang, "Digital predistortion of wideband power amplifier with single undersampling adc," *IEEE Microwave and Wireless Components Letters*, vol. 27, no. 11, pp. 1016–1018, 2017.

[20] A. Katz, J. Wood, and D. Chokola, "The evolution of pa linearization: From classic feedforward and feedback through analog and digital predistortion," *IEEE Microwave Magazine*, vol. 17, no. 2, pp. 32–40, 2016.

[21] P. L. Gilabert, G. Montoro, and E. Bertran, "Fpga implementation of a real-time narma-based digital adaptive predistorter," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 7, pp. 402–406, 2011.

[22] D. Painchaud, Z. Sun, and M. F. Schemmann, *Predistortion using fpga*, US Patent 8,145,066, 2012.

[23] Y. Ma, Y. Yamao, Y. Akaiwa, and C. Yu, "Fpga implementation of adaptive digital predistorter with fast convergence rate and low complexity for multi-channel transmitters," *IEEE Transactions on Microwave Theory and Techniques*, vol. 61, no. 11, pp. 3961–3973, 2013.

[24] *Fpga vs microcontroller - advantages of using an fpga*, 2019.

[25] F. Mkadem and S. Boumaiza, "Physically inspired neural network model for rf power amplifier behavioral modeling and digital predistortion," *IEEE Transactions on Microwave Theory and Techniques*, vol. 59, no. 4, pp. 913–923, 2011.

[26] X. Feng, Y. Wang, B. Feuvrie, A.-S. Descamps, Y. Ding, and Z. Yu, "Analysis on lut based digital predistortion using direct learning architecture for linearizing power amplifiers," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 132, 2016.

[27] L.-K. Ting, R. Woods, and C. F. Cowan, "Virtex fpga implementation of a pipelined adaptive lms predictor for electronic support measures receivers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 1, pp. 86–95, 2005.

[28] G. Golub, "Numerical methods for solving linear least squares problems," *Numerische Mathematik*, vol. 7, no. 3, pp. 206–216, 1965.

[29] R. Raich, H. Qian, and G. T. Zhou, "Orthogonal polynomials for power amplifier modeling and predistorter design," *IEEE transactions on vehicular technology*, vol. 53, no. 5, pp. 1468–1479, 2004.

[30] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Springer, 2010, pp. 177–186.

[31] S. S., *Gradient descent: All you need to know*, 2018.

[32] D. Brandwood, "A complex gradient operator and its application in adaptive array theory," in *IEE Proceedings H-Microwaves, Optics and Antennas*, IET, vol. 130, 1983, pp. 11–16.

[33] S. S. Haykin, *Adaptive filter theory*, 2nd ed. Prentice-Hall, 1986.

[34] M. Sun, T. Xu, H. Guo, and H. Zhong, "Wideband wireless transmitter identification based on hammerstein-wiener model," *Journal of Applied Science and Engineering*, vol. 21, no. 2, 261269, 2018.

[35] S. Hafsi, K. Laabidi, and M. Ksouri-Lahmari, “Identification of wiener-hammerstein model with multisegment piecewise-linear characteristic,” in *2012 16th IEEE Mediterranean Electrotechnical Conference*, IEEE, 2012, pp. 5–10.